

AD-A148 348

COSAGE (CONCEPTS ANALYSIS AGENCY'S COMBAT SAMPLE  
GENERATOR) ANALYSIS AND. (U) SCIENCE APPLICATIONS INC  
LA JOLLA CA D A HEINBURGER ET AL. 29 APR 84

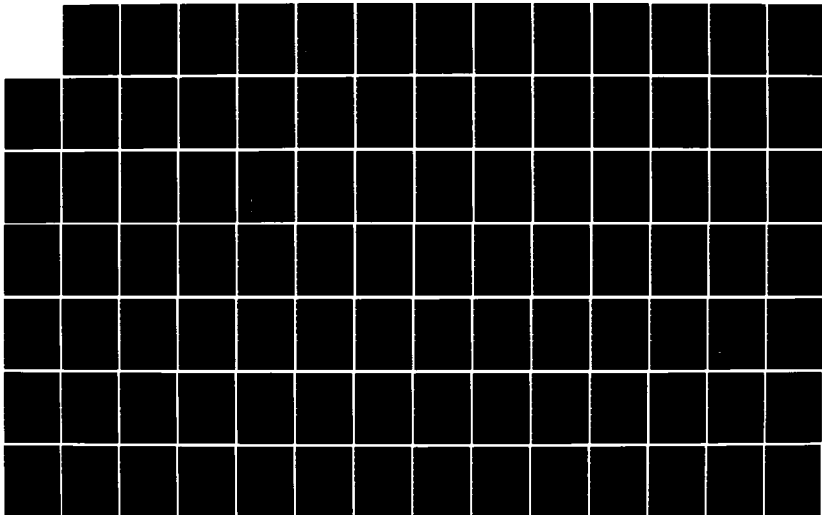
1/2

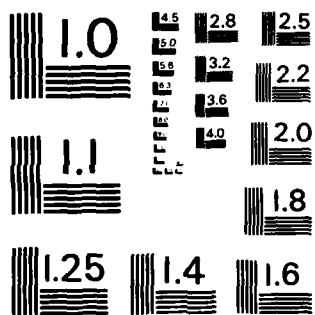
UNCLASSIFIED

MDA903-83-C-0424

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

①

AD-A148 348

FINAL DESIGN REPORT  
FOR THE  
STUDY ENTITLED  
"COSAGE ANALYSIS AND DESIGN REPORT"  
Volume I

science applications, inc.

DTIC FILE COPY

DTIC  
UNCLASSIFIED  
DEC 0 1984

THIS CONTAINS INFORMATION  
OF PUBLIC CONCERN  
DISTRIBUTION IS UNLIMITED

84 11 20 176

SCIENCE APPLICATIONS, INC.

FINAL DESIGN REPORT  
FOR THE  
STUDY ENTITLED  
"COSAGE ANALYSIS AND DESIGN REPORT"  
Volume I

Contract No.  
MDA903-83-C-0424

Contract Expiration Date:  
April 29, 1984

Prepared for:  
U.S. Army - Concepts Analysis Agency  
Bethesda, MD 20014  
Mr. Hugh Jones

Prepared by:  
Science Applications, Inc.  
La Jolla, CA 92038  
Mr. Donald A. Heimbürger  
Ms. Marcia A. Metcalfe  
Ms. Suellen S. Worrells  
Ms. Diane K. Graham

DTIC  
ELECTE  
DEC 0 1984  
E

SAI

This document has been approved  
for public release and its  
distribution is unlimited.

TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
1.0 INTRODUCTION	1-1
2.0 ANALYSIS TOOLS & TECHNIQUES	2-1
2.1 Source Code Instrumentation	2-2
2.2 VAX System Performance Monitoring Tool	2-2
2.3 Metrics Analysis	2-3
2.4 VAX SIMSCRIPT Compiler Error Checking	2-4
3.0 ANALYSES PERFORMED AND RESULTS OBTAINED	3-1
3.1 Analysis of COSAGE Model Invocations	3-1
3.2 Analysis of COSAGE Model CPU Usage	3-5
3.3 Analysis of COSAGE Model Execution	3-8
3.4 Analysis of COSAGE Model SIMSCRIPT Execution	3-13
3.4.1 Anomalies which Occurred While Reading the Input Data	3-13
3.4.2 Anomalies Which Occurred During Simulated Time	3-15
3.5 Metrics Analysis	3-17
3.5.1 Control Complexity Metric	3-17
3.5.2 Operand Complexity Metric	3-42
4.0 RECOMMENDED CHANGES	4-1
4.1 Exponentiation	4-1



4.2 Inefficient Mathematical Expressions	4-2
4.3 Unnecessary SQRT.F Usage	4-3
4.4 Schedules/Reschedules	4-3
4.5 Removal/Replacement of Identified Modules	4-4
4.6 Utilize SIMSCRIPT Text Feature	4-7
4.7 Perform Thorough Analysis of the 26 Most Frequently Invoked Modules	4-11
4.8 Modularize Candidate Processes	4-13
4.9 Standardize the COSAGE Source Code	4-20
4.10 Develop Graphical Input/Output Capabilities	4-21
5.0 PROPOSED PREAMBLE	5-1
5.1 Existing Structure	5-1
5.2 Proposed Structure	5-1
6.0 SUMMARY	6-1

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
<i>form 50 per</i>	
Distribution/	
Codes	
Number	
Date	
<b>A-1</b>	



**SAI**

LIST OF FIGURES

<u>FIGURE</u>	<u>PAGE</u>
3.1 COSAGE Summary Invocation Report	3-3
3.2 COSAGE Hourly Invocation Summary Report	3-4
3.3 COSAGE CPU Usage Summary	3-6
3.4 Cosage Invocation and CPU Summary	3-7
3.5 SPM Results of COSAGE Image Region	3-10
3.6 Configuration Control Form Used for Metrics	3-18
3.7 Level of IF Nesting	3-19
3.8 Modules Ranked by IF Tests	3-20
3.9 Modules Ranked by Functional IF Tests	3-28
3.10 Modules Ranked by Maximum IF Depth	3-35
3.11 Counting Operators for the Halstead Length Metric	3-43
3.12 Counting Operands for the Halstead Length Metric	3-45
3.13 Modules Ranked by Halstead Length	3-46
4.1 Schedule Testcase	4-5
4.2 Reschedule Testcase	4-6
4.3 344,157 Call Statements Testcase	4-9
4.4 344,157 Assignment Statements Testcase	4-10
4.5 Current Function ACT.RANGE	4-12
4.6 Proposed Function ACT.RANGE	4-14



4.7 Current Routine RANGE.COMPUTE	4-15
4.8 Proposed Routine RANGE.COMPUTE	4-16
4.9 Example of High-level Comment System and Code Standardization	4-18
4.10 Current COSAGE Routine	4-22
5.1 Existing PREAMBLE Scheme	5-2
5.2 Recommended Hierarchical PREAMBLE Scheme	5-3





LIST OF TABLES

<u>TABLE</u>	<u>PAGE</u>
4.1 Modules to be Deleted	4-8



## 1.0 INTRODUCTION

Science Applications, Inc. (SAI) has conducted a study of Concepts Analysis Agency's Combat Sample Generator (COSAGE) Program. This program consists of over 30,000 lines of SIMSCRIPT source code. It requires approximately 1.5 hours of SPERRY 1100/83 CPU time to execute and the maximum amount of static memory available (262K words). The goal of this study is to identify fruitful areas for COSAGE optimization which will reduce the COSAGE memory requirement as well as the execution time. To accomplish this, SAI has performed static and dynamic analyses of the source code. The purpose of this report is threefold:

- 1. To present the results of the dynamic analyses effort;
- 2. To preview the recommended changes; and
- 3. To provide suggested COSAGE model PREAMBLE revisions.

This report is presented in three (3) volumes. The remainder of Volume I is organized in five sections:

- 1. Section 2.0 presents the tools and techniques which were utilized to perform the dynamic analyses.
- 2. Section 3.0 discusses the dynamic analyses performed and the results obtained.
- 3. Section 4.0 previews the recommended optimization changes.
- 4. Section 5.0 contains revision recommendations for the COSAGE model PREAMBLE.



- Section ~~8.0~~<sup>6</sup> provides a summary of the optimization effort. ←

Volume II is the COSAGE SIMSCRIPT source code for the VAX computer which has been processed by SAI-SDDL\*; Volume III contains COSAGE Hourly Invocation Reports for random number seeds 3, 6, and 10, respectively.

\* A trademark of Science Applications, Inc.



## 2.0 ANALYSIS TOOLS AND TECHNIQUES

To facilitate the required analyses, SAI rehosted the COSAGE model on a virtual memory VAX computer in a "test suite" environment. This "test suite" incorporates numerous software tools and techniques:

- Science Applications, Inc.'s Software Design and Documentation Language (SAI-SDDL) was used to format the COSAGE source code and provide automated summaries such as a table of contents, module invocation hierarchy tree, and a variety of cross-reference listings. SAI-SDDL was also used for developing COSAGE input format specifications
- System Performance Monitoring (SPM) Tool was utilized to analyze COSAGE model execution at the operating system level
- Metrics were applied to obtain quantitative assessments of the complexity of the COSAGE source code
- VAX SIMSCRIPT Compiler was used to identify source code anomalies which the SPERRY compiler is unable to detect.



The remainder of this section discusses in more detail the tools and techniques which were utilized for the dynamic analyses.

## 2.1 Source Code Instrumentation

SAI has instrumented the COSAGE model source code in order to identify areas that would most benefit from optimization (i.e., routines most frequently invoked during model execution as well as COSAGE CPU usage by simulated hour). In order to capture routine invocations, counters were inserted into every COSAGE routine/process/event. These counters were incremented each time the module was invoked. Additionally, an event was developed that writes the counter values to a data file on an hourly (simulated time) basis and then clears the counters for the next data collection period. CPU usage was determined by utilizing appropriate VAX system routines. In addition, the event mentioned above was modified to write the CPU usage for each simulated hour to a data file.

## 2.2 VAX System Performance Monitoring (SPM) Tool

SAI analysts applied the SPM tool to the COSAGE model. VAX-11 SPM is a set of programs which collect and report performance statistics for VAX/VMS systems. General performance statistics can be collected on a system-wide basis, and detailed statistics can be collected on a per-process basis.

Included in the SPM set is a package for measuring where a user's program is spending its time. To do so, the package periodically samples the program counter of the running program, determines in which portion/routine of the program each such sample falls, and displays the resulting information in histogram



form. Program counter samples are collected by trapping a clock interrupt every 10 milliseconds. The user is able to specify how the program is to be divided into sections called buckets for performance data collection. A bucket is defined by an address range, and accumulates the number of program samples in that address range through the use of a counter. The structure of the program to be measured may be specified in terms of very large divisions or individual routines as well as starting and ending addresses.

### 2.3 Metrics Analysis

SAI has employed two metrics analysis techniques with the COSAGE model. The first metric, control complexity, was developed by McCabe (Ref. [1], Appendix A, Volume III) and identifies software modules that are difficult to test and maintain. Control complexity is measured by cyclomatic number, which is the number of independent paths through the code. The criterion value for cyclomatic number is usually 10. That is, if there are more than 10 independent paths in a routine, then it is usually not possible to fully test all paths. Consequently, the program reliability and maintainability could be adversely affected.

The second metric, operand complexity, is traditionally measured by Halstead's length metric (Ref. [2], Appendix A, Volume III) which is the sum of the operator occurrences (e.g., +, -, \*, /, >, < =, ≠, \*\*, ADD, SUBTRACT) and operand occurrences (e.g., variables, attributes, entities, sets).

Typically, if the Halstead length metric is 270 or above per routine, it is indicative of poor design practices during the module/submodule allocations (modularization phase). It has also



been correlated with other measures such as number of bugs in a program, required programming/reprogramming time, and the quality of programs (Ref. [3], Appendix A, Volume III).

#### 2.4 VAX SIMSCRIPT Compiler Error Checking

SAI re-hosted the COSAGE model on a VAX computer to perform the required analyses for a variety of reasons. One major consideration was the upgraded SIMSCRIPT compiler features which are implemented in the VAX computer version and not currently available in the SPERRY computer compiler. The VAX SIMSCRIPT enhancements include:

- Checking for subscripts out of bounds to an array, permanent entity, or temporary entity
- Identifying references to a temporary attribute or an array element of a quantity that has been destroyed or released
- Verifying that the number of words for arguments agree in definition and use
- Mode checking



### 3.0 ANALYSES PERFORMED AND RESULTS OBTAINED

Numerous analyses were performed by SAI. This section discusses these analyses and presents the results obtained.

#### 3.1 Analysis Of COSAGE Model Invocations

In order to capture the number of invocations for each COSAGE source code routine, an "ADD" statement was inserted as the first executable statement in each routine. These statements increment an array element associated with a particular routine each time the routine is executed. The array (ANAL.CTR) was defined in the COSAGE PREAMBLE; it was dimensioned by the number of routines in the source code.

In order to report the number of invocations per hour of simulated time, an event was written and added to the COSAGE model that writes to a disk file the name of each routine and the number of invocations recorded per simulated hour. It then clears the counter array and reschedules itself to execute in one simulated hour.

In order to increase the useability of the data gathered in this manner, a formatting postprocessor was written which ranks the routines by highest number of invocations. For any user-specified number of routines in the COSAGE model (e.g., top 10, top 50, all), the number of invocations, the percent of hourly





calls, and an accumulated hourly percent of calls for each hour of simulated time is printed. Appendix B (Volume III) contains the output of the postprocessor when all 264 modules were requested using random number seed 3.

In addition, a summary report is produced at the end of the simulation. This COSAGE summary invocation report ranks the selected number of routines, giving the number of invocations for each, the percent of total calls, and the accumulated total percentage. Figure 3.1 presents this summary report.

A second summary report shows the number of invocations per hour of simulated time and the percent of total invocations as a number and as a line on a bar chart. Figure 3.2 presents this hourly invocation summary.

Analysis of this output has helped to direct and focus the optimization investigation. It is clear from the results of Figure 3.1 that 10% (26) of the COSAGE modules account for over 93% of all module invocations and should be closely scrutinized. Seven of the 26 were already noted for optimization with the \OPTIMIZE token during the static analysis, and one was marked as a deletion candidate. The two processes, ASSESSMENT and SHOOTOUT, were both in the largest dozen modules ranked by source lines.

Figure 3.1 also reveals two closely coupled sets of program modules. The routines JOHNSON.CRITERIA, PROB.INF, PROB.TIME, and SEARCH were each invoked 344,157 times, accounting for over 20% of all invocations; MRT.TO.FREQ and TEMPERATURE.ATTENUATION were each invoked 75,923 times. These algorithms and their interfaces should be streamlined to minimize the overhead of the invocations themselves.



# SCIENCE APPLICATIONS, INC.

## COSAGE SUMMARY INVOCATION REPORT

TOP 26 (10%) INVOKED ROUTINES		TOTAL INVOCATIONS	PCT CALLS	ACC TOTAL PCT
1	FUNCTION_ACT.RANGE	1189098	17.459	17.459
2	ROUTINE_RANGE.COMPUTE	792643	11.638	29.097
3	ROUTINE_PK.COMPUTE	741236	10.883	39.980
4	ROUTINE_PROX.CHECK	399966	5.872	45.852
5	ROUTINE_JOHNSON.CRITERIA	344157	5.053	50.906
6	ROUTINE_PROB.INF	344157	5.053	55.959
7	ROUTINE_PROB.TIME	344157	5.053	61.012
8	ROUTINE_SEARCH	344157	5.053	66.065
9	ROUTINE_TIME.TO.DETECT	312629	4.590	70.655
10	ROUTINE_FRAC.COMPUTE	291000	4.273	74.927
11	ROUTINE_CONTRAST.TO.FREQ	268234	3.938	78.866
12	ROUTINE_LOCATE.SECTOR	142090	2.086	80.952
13	ROUTINE_CHECK.ENGAGEMENT	129648	1.904	82.856
14	ROUTINE_SIZE.ESTIMATE	128398	1.885	84.741
15	ROUTINE_MRT.TO.FREQ	75923	1.115	85.855
16	ROUTINE_TEMPERATURE.ATTENUATION	75923	1.115	86.970
17	ROUTINE_FINAL.COVERAGE	74273	1.091	88.061
18	PROCESS_ASSESSMENT	53613	.787	88.848
19	ROUTINE_PDB.DETECTION	44444	.653	89.500
20	FUNCTION_COMBINATIONS	41320	.607	90.107
21	ROUTINE_DEQ.FEBA.SET	40041	.588	90.695
22	ROUTINE_ENQ.FEBA.SET	39866	.585	91.280
23	PROCESS_SHOOT.OUT	36804	.540	91.821
24	EVENT_PDB.ACTIVATION	35159	.516	92.337
25	ROUTINE_WEIBULL.F	23942	.352	92.688
26	FUNCTION_EST.RANGE	23356	.343	93.031
TOTAL INVOCATIONS =		6810855		

Figure 3.1

COSAGE Summary Invocation Report



PAGE 151

COSAGE INVOCATION SUMMARY

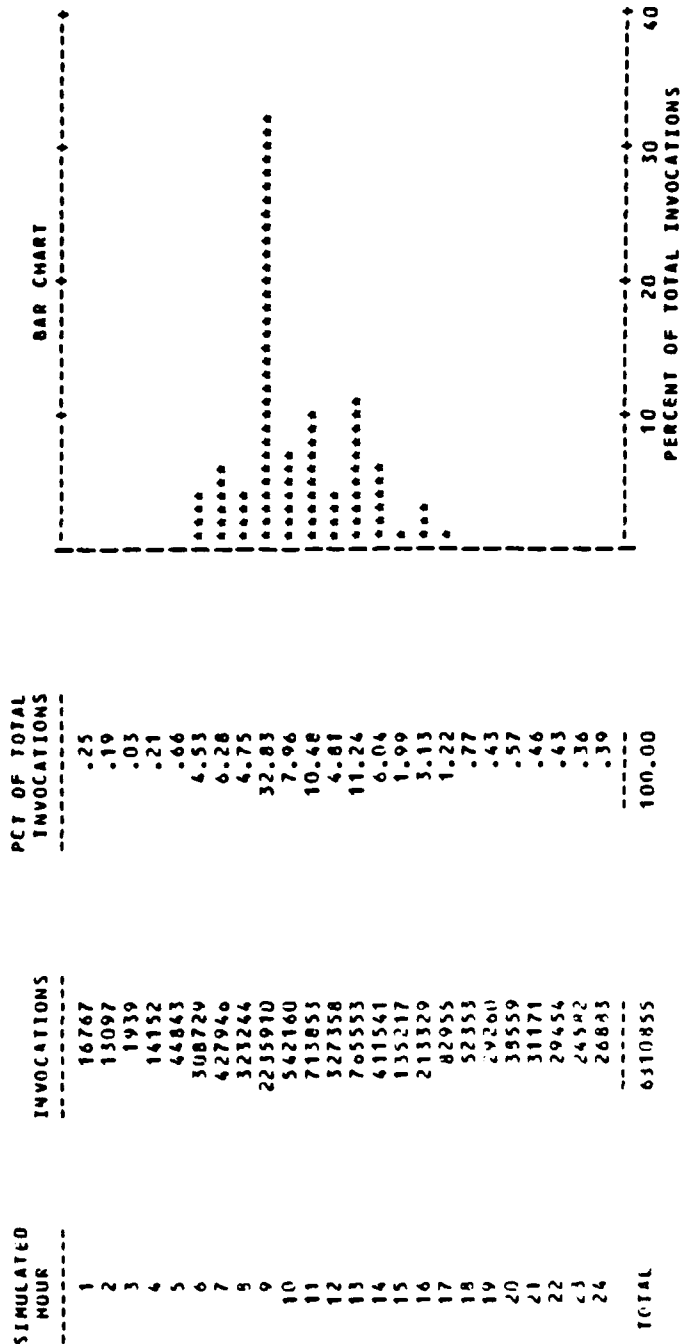


Figure 3.2

COSAGE Hourly Invocation Summary Report



Finally, infrequently used routines may be eliminated, thereby reducing the overall size of the COSAGE program. Appendix B, Volume III, provides a good departure point to purge the program.

It should be noted that the analyses performed and results obtained in this section are based on executing the COSAGE model using SIMSCRIPT's random number seed 3. However, SAI analysts also conducted analyses using two additional random number seeds; namely, 6 and 10. The analyses results for random number seed 6 are included in Appendix C, Volume III; the results from seed 10 are in Appendix D, Volume III.

### 3.2 Analysis of COSAGE Model CPU Usage

An additional analysis was performed by instrumenting the COSAGE source code. This analysis yielded CPU usage by simulated hour. To ascertain this information, LIB\$INIT\_TIMER was invoked during the COSAGE initialization phase. This routine initialized the VAX system timing mechanism. Then, LIB\$STAT\_TIMER (another VAX system routine) was called after each hour of simulated time. This was accomplished by modifying the event which was written to capture the number of invocations. The change caused the hourly CPU usage data to be written to a data file. Additionally, the postprocessor which was developed to produce the COSAGE Hourly Invocation Report was enhanced to present hourly CPU usage. A sample COSAGE CPU Usage Summary report is shown in Figure 3.3. The next step involved integrating the results of the COSAGE Hourly Invocation Summary report and the COSAGE CPU Usage Summary report into a single summary. A sample COSAGE Invocation and CPU Usage Summary is shown in Figure 3.4.



COSAGE CPU USAGE SUMMARY

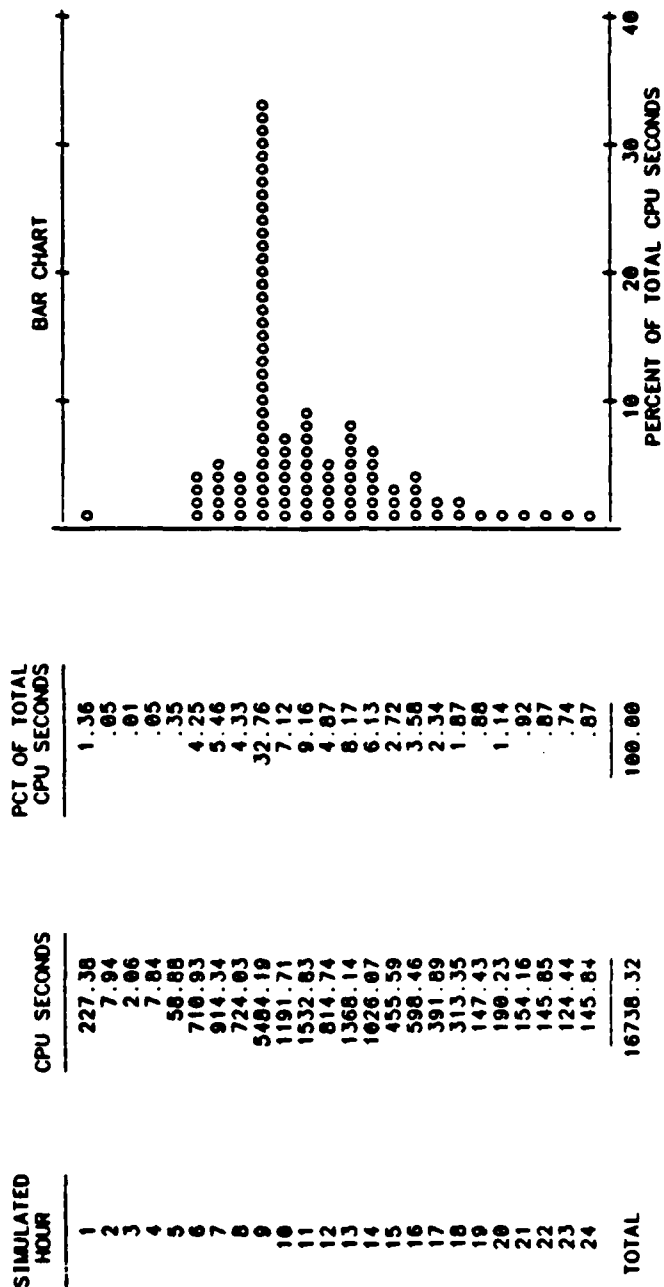


Figure 3.3  
COSAGE CPU Usage Summary

COSAGE INVOCATION AND CPU USAGE SUMMARY

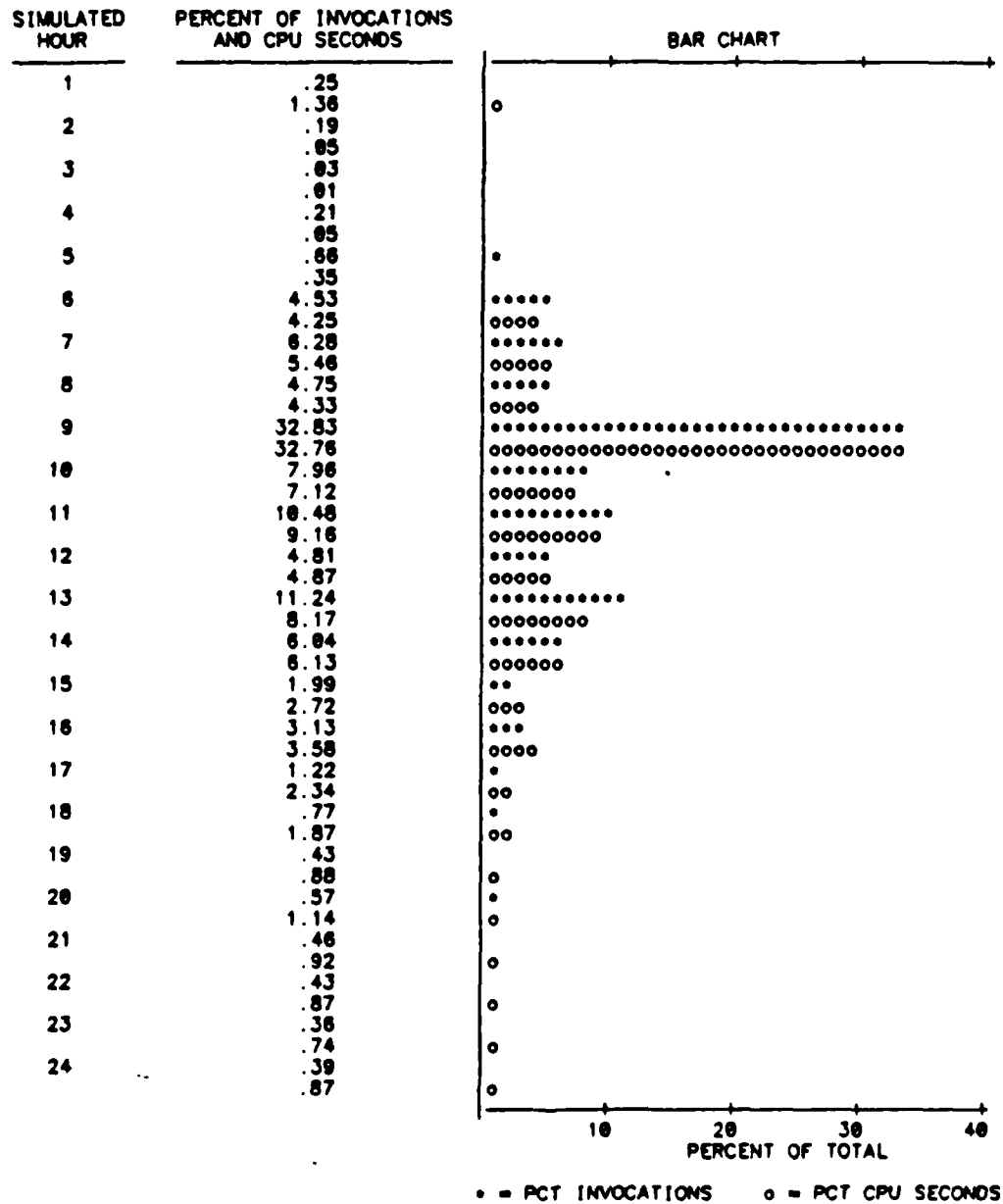


Figure 3.4  
COSAGE Invocation and CPU Usage Summary



### 3.3 Analysis Of COSAGE Model Execution

SAI instrumented the COSAGE model environment with the System Performance Monitoring (SPM) tool to gather samples of the model counter. This was done to determine where the program was spending its time.

In order to avoid modifying the COSAGE program itself, the executable image was linked with the SPM module IMGSHLL specified as the DEBUG option. The IMGSHLL module is a program which automatically starts and stops the sampling routines. When linked this way, IMGSHLL is invoked by the VMS operating system as if it were the debugger. It thus gets control before the user program. This allows it to initiate clock sampling before starting the user program and to terminate the sampling after the user program exits. The program counter samples are taken every 10 milliseconds and accumulated in a file. Upon completion of a COSAGE execution, the file containing program counter samples can then be used in the analysis.

The next step is to define address ranges of interest. The program, and its associated address space, was divided into smaller units. This was done by specifying five primary areas. These included operating system, program control region, COSAGE image region, user program region at addresses above the COSAGE image, and the SIMSCRIPT library. The addresses were set up for the COSAGE image region so that each program module would be accounted for individually. The SPM module IMGDEFINE was executed; it specifies how the program is to be broken into address buckets for data collection. The output of the IMGDEFINE



module is a single file containing all necessary information about how the user has divided the program into buckets or address ranges. This is an empty bucket file and is ready to be used along with the sampling output from a COSAGE execution.

The sampling output consists of a file produced by clock-driven traps which collect program counter values. This file, along with the empty bucket file generated by the IMGDEFINE module, is then used as input to the IMGREPORT module of SPM. IMGREPORT tallies the program counter samples in the appropriate buckets and produces a histogram showing the number of tallies in each bucket.

The results in the histogram are shown as percentages. For this analysis, the results were as follows:

Operating system	0.00%
Program Control region	0.50%
COSAGE Image region	28.93%
User region above COSAGE	10.36%
<u>SIMSCRIPT Library</u>	<u>80.18%</u>
	99.97%

Of the 28.93% of samplings which were attributed to the COSAGE Image region, the individual routines trapped and their relative percentages are shown in Figure 3.5. These routines, when summed, account for 28.77% of the samplings in the COSAGE Image region. The difference can be attributed to precision of the SPM package which rounds to the nearest one-hundredth of one percent.

There were a total of 1,703,991 samples taken; the percent in defined buckets was 99.97%, and the number of address ranges represented was 535. The program control region represents





# SCIENCE APPLICATIONS, INC.

5.87% RTIME.TO.DETECT  
4.74% RFRAC.COMPUTE  
4.00% RPK.COMPUTE  
2.11% RSHOOT.OUT  
1.34% RFINAL.COVERAGE  
0.81% RACT.RANGE  
0.74% RRANGE.COMPUTE  
0.73% RTARGET.ANALYSIS  
0.51% RPROB.INF  
0.44% RCONTRAST.TO.FREQ  
0.44% RPROB.TIME  
0.43% RLOCATE.SECTOR  
0.42% RPROX.CHECK  
0.41% RSEARCH  
0.33% RDEQ.FEBA.SET  
0.30% RBTRY.EFFECTS  
0.30% RNOISE.DEGRADE  
0.23% RASSESSMENT  
0.22% RLOS.CHECK  
0.21% RFORWARD.OBSERVER  
0.19% HTIME.R  
0.19% RSIZE.ESTIMATE  
0.18% RCHECK.ENGAGEMENT  
0.15% RUNIT.INPUT  
0.14% TSS.SET  
0.13% RPDB.DETECTION  
0.13% RPDB.ACTIVATION  
0.13% RFIREF.MISSION  
0.12% RTEMPERATURE.ATTE  
0.12% ROUTPUT.ATTRITION  
0.11% RCHECK.PROX  
0.11% RHE.WLA  
0.11% RJOHNSON.CRITERIA  
0.11% REST.COVERAGE  
0.11% RFA.BN.ASGN  
0.10% RDUST.EFFECTS  
0.09% RCHK.FD.TR  
0.09% RNEW.SEGMENT  
0.09% RFO.DETECTION  
0.07% RMIN.MOVE  
0.07% RVOLLEY  
0.06% RCFR.ACTIVATION  
0.06% RUPDATE.LOC  
0.06% RENQ.FEBA.SET  
0.05% RTARGET.REPORT  
0.05% RFA.BN.MOVEMENT  
0.05% RENGAGEMENT  
0.05% RHE.OR.ICM.COMPUT  
0.04% RMRT.TO.FREQ  
0.04% TFO.CAND.DET.LIST  
0.04% RCOMBINATIONS  
0.04% RFEB.A.BAND  
0.04% RCHANGE.LOC  
0.04% RWEIGHTED.VOLLEYS  
0.04% HUNIFORM.F

Figure 3.5  
SPM Results of CUSAGE Image Region



SCIENCE APPLICATIONS, INC.

0.04% RICM.WLA  
0.04% HNORMAL.F  
0.03% ZSS.SET  
0.03% REST.RANGE  
0.03% RMARGINAL.EFFECTS  
0.03% RCFR.DEGRADE  
0.03% REMPTY  
0.02% DPDB.ACTIVATION  
0.02% HGAMMA.F  
0.02% RLINE.OF.SIGHT  
0.02% USO.LIST  
0.02% RCFR.DETECTION  
0.02% RHE.LA.INPUT  
0.02% RBLOCK.LOS  
0.02% TPDB.OP.Q  
0.02% HWEIBULL.F  
0.02% TFD.TR.QUEUE  
0.02% HBINOMIAL.F  
0.02% HCOMPUTE.D  
0.01% ZIF.RATE.LIST  
0.01% RCFR.OPERATOR  
0.01% RSTART.BATTLE  
0.01% RLOCATE.SEARCH.AR  
0.01% DIF.VOLLEY  
0.01% ZUE.TARGET.LIST  
0.01% ZFO.CAND.DET.LIST  
0.01% RSTOP.ARTY.MOVEME  
0.01% HRANDI.F  
0.01% ROUTPUT.EXPENDITU  
0.01% DFIRE.MISSION  
0.01% RGENERAL.BATTLE  
0.01% RCHK.COMP.TR  
0.01% ZBY.FM.QUEUE  
0.01% RSTART.MOVE  
0.01% RBTRY.FM.DEQ  
0.01% RREM.EFFECTS.COMP  
0.01% UIF.RATE.LIST  
0.01% DFO.DET.CANDIDATE  
0.01% RCOMPARE.TRS  
0.01% RCLEAN.UP.FIRE.MI  
0.01% RPOSITION  
0.01% DFIRING.TABLE  
0.01% ZSO.LIST  
0.01% RCHECK.FORCE  
0.01% RCREATE.FORCE  
0.01% RSWITCH.FO  
0.01% XUN.SEGMENT.LIST  
0.01% RWITH.DRAW  
0.01% RARTY.ASSESS  
0.01% RSTART.ARTY.MOVEM  
0.01% RGET.TERRAIN  
0.01% XSO.LIST  
0.01% UUN.LOS.LIST  
0.01% TBY.FM.QUEUE  
0.01% RBTRY.INPUT

Figure 3.5

SPM Results of CUSAGE Image Region

(continued)



0.01% DUPDATE.LOC  
0.01% RFINISH.COMPUTATI  
0.01% TBN.CAN.FM.SET  
0.01% RFDC.TR.DEQ  
0.01% XFO.CAND.DET.LIST  
0.01% UUE.TARGET.LIST  
0.01% DTARGET.REPORT  
0.01% RSENSOR.INPUT  
0.01% UUN.SEGMENT.LIST  
0.01% RUNIT.ENVIR  
0.01% DSHOOT.OUT  
0.01% RPK.INPUT  
0.01% RUNIT.ASSIGNMENT  
0.01% RSEGMENT.ADJUST  
0.01% RPDB.OPERATOR  
0.01% RKV.PRINT

Figure 3.5

SPM Results of COSAGE Image Region

(continued)

activities performed by the system on behalf of the image such as user stack usage and image input/output. The user region above COSAGE represents the operating system and the debugger. Any discrepancies between percentages contained in the report and shown in the total may be attributed to round-off. The remaining .03% of activity not accounted for was in an address range which was not requested in this analysis.

### 3.4 Analysis Of COSAGE Model SIMSCRIPT Execution

The SIMSCRIPT compiler on the VAX computer incorporates language enhancements which are not available on the SPERRY computer. These features made it possible to identify anomalies which heretofore had gone undetected. Anomalies have been grouped into two categories: ones that occurred while reading the input data and ones that occurred during simulated time. These irregularities are discussed further below.

#### 3.4.1 Anomalies Which Occurred While Reading the Input Data

In the course of implementing the COSAGE model on the VAX computer, a number of problems were encountered while reading the data file provided. Each problem and solution is listed below.

1. Problem: Need explicit unit number for input file.

Solution: Opened unit 4 in new module OPEN.INPUT.OUTPUT.FILES for reading input data.

2. Problem: Divide by zero in SYS.INPUT. Customer provided information that the data items for NUM.POSITION.REPORT and CLP.ON were reversed.

Solution: Corrected order of the data items in the input file.

3. Problem: Unreserved array in PK.INPUT. PK.F.MOV.FAC does not seem to be allocated automatically.

Solution: Reserved array explicitly.

4. Problem: Subscript out of range in CAT.TU.INPUT. Data originally read with ALPHA 6 format and now being read as TEXT requiring a blank space in data.

Solution: Inserted a blank space in data item.

5. Problem: Not sufficient virtual address due to size of model.

Solution: Wrote macro routine to increase MAX VIRTUALADDR to 3 megabytes.

6. Problem: Zero entity pointer or unreserved array in BRTY.INPUT. BRTY's 37 through 40 did not have proper equipment.

Solution: Added LART1 equipment to units 204, 205, 206 and 207.

7. Problem: Subscript out of range in SENSOR.INPUT. When SENSOR.TYPE is 1 and ST.NAME is "FO", SENSOR.MODEL must be less than 10 or subscript goes out of range.

Solution: Changed data so that SENSOR.MODEL is 1 for those cases.



8. Problem: Argument passed to H.SIGN.F must be real (called from SENSOR.INPUT).

Solution: Explicitly defined DISTANCE as a real variable in SENSOR.INPUT.

9. Problem: Subscript out of range in SENSOR.INPUT. (Problem same as 7. above).

Solution: Changed data so that SENSOR.MODEL is 1 for those cases.

10. Problem: Invalid character in I format in MADS.INPUT. .NUM.RH read in this routine was incorrect in many instances. It is being used as a loop counter for subsequent reads and must correspond to the number of data items following.

Solution: Determined correct values for .NUM.RH and replaced original incorrect values in the data.

#### 3.4.2 Anomalies Which Occurred During Simulated Time

After the COSAGE program read all the input data and scheduled the initial events and processes, a START SIMULATION statement was executed. From this point to normal execution termination, the SIMSCRIPT compiler-generated timing routine, TIME.R, directed the execution of the program. The timing routine updated the simulated time, TIME.V, and invoked the subroutines corresponding to the required event or process.

As the program executed new paths, or repeated previously executed paths with new data, a variety of SIMSCRIPT execution errors were encountered. A complete list of the

execution problems and the solutions applied to continue execution is contained in the accompanying source code (Volume II). The module entitled PROGRAM CHANGES on page 2 matches a token of the form CHG\NN, where NN is a 2-digit number, with the location(s) in the code which was affected. These changes, while numerous and labor-intensive to implement, resulted from several broad categories of problems. These categories included the following:

- Compiler Variations - These included both VAX and SPERRY implementation idiosyncrocies.
- Zero Subscript Error - There was a wide variety of reasons for the subscript being zero, with misspellings, attributes used but not initialized, and faulty logic leading the list.
- Reference to a Destroyed Entity - A reference to a destroyed entity resulted from an attempt to retrieve data about an entity or process after it had exited from the simulation. The solutions usually required obtaining the data before the entity was destroyed or zeroing-out the pointer that referred to the entity.
- Precision differences - Since a real variable on the VAX defaults to 64 bits (vs 36 on the SPERRY), some differences occurred based on the extended precision and round-off.
- Number and Mode Mismatches for Arguments - Some calls to subroutines contained less than the specified number of arguments; those calls were supplemented to fulfill the list. Some calls specified arguments in a mode

different from that specified in the called routine; those differences were resolved.

- Subscript Out of Range - These almost always were a result of faulty logic.
- Division By Zero - The rare cases where this occurred were tested for and handled as exceptions.

### 3.5 Metrics Analysis

Two metric analyses were performed on the COSAGE model: the control complexity metric and the operand complexity metric. The details of these two measures and the results are presented below.

#### 3.5.1 Control Complexity Metric

In order to determine the control complexity metric, (number of paths through the code), each COSAGE source routine was examined for the number of IF tests performed. A separate count was kept of the number of IF tests that controlled debug output and the maximum depth of IF test nesting within the routine. This information was tallied using an SAI-developed configuration control form. A sample form is shown in Figure 3.6. Figure 3.7 is provided to illustrate the procedure employed to glean this measure. As can be seen, this section of code has eight IF tests, none of which control debug output. It also has a maximum depth of nesting of four (IFs 2, 3, 5, 6).

A post-processor was written to tabulate the data gathered in this manner and to produce three reports. The first report lists the modules ranked by IF tests (see Figure 3.8). The





# SCIENCE APPLICATIONS, INC.

		ROUTINE NAME	DIFF	NO	NO
51	2895	ROUTINE NAME			
52	2896	ROUTINE NAME	T001	1	1
53	2897	ROUTINE NAME	T002	1	1
54	2901	ROUTINE NAME	T003	1	1
55	2920	ROUTINE CREATE, SDCS	T004	1	1
57	2947	ROUTINE CREATE, SDCS	T005	1	1
58	3113	ROUTINE REBA, INITIAL	T006	1	1
60	3088	ROUTINE FILE, AC, SDCS	T007	1	1
62	3183	ROUTINE FILE, KAP, SDCS	T008	1	1
63	3242	ROUTINE ACUM, TP, LIST	T009	1	1
64	3273	ROUTINE GENERAL, SDCS	T010	1	1
68	3458	ROUTINE ORIENTATION	T011	1	1
72	3646	ROUTINE UNIT, ASSIGNMENT	T012	1	1
73	3693	TITLE MOVEMENT/TERRAIN ROUTINES			
76	3700	ROUTINE ADJUST	M001	1	1
77	3823	ROUTINE BLOCK, LOS	M002	1	1
80	3963	ROUTINE CHANGE, LOC	M003	1	1
84	4123	ROUTINE ENG, MOVE	M004	1	1
86	4212	ROUTINE AS, EN, MOVEMENT	M005	1	1
90	4424	ROUTINE INITIAL, DETEC	M006	1	1
91	4425	ROUTINE INITIAL, MOVE	M007	1	1
92	4446	ROUTINE INITIAL, DETEC	M008	1	1
93	4473	ROUTINE LINE, CP, SIGHT	M009	1	1
95	4575	ROUTINE LOCATE, SEARCH, SDCS	M010	1	1
97	4634	ROUTINE LOCATE, SEARCH	M011	1	1
98	4678	ROUTINE LOS, CHECK	M012	1	1
100	4735	ROUTINE ENG, DELAY	M013	1	1
101	4803	ROUTINE ENG, MOVE	M014	1	1
104	4920	ROUTINE NEW, SEGMENT	M015	1	1
107	5037	ROUTINE SEARCH	M016	1	1
108	5101	ROUTINE SEARCH, SDCS	M017	1	1
109	5147	ROUTINE SEARCH, LIST	M018	1	1
110	5206	ROUTINE SEARCH, SDCS	M019	1	1
112	5243	ROUTINE SEARCH, CHECK	M020	1	1
116	5341	ROUTINE SEARCH, SDCS	M021	1	1
118	5344	ROUTINE SEARCH, MOVE	M022	1	1
119	5403	ROUTINE SEARCH, ADJUST	M023	1	1
117	5453	ROUTINE SEARCH, TC, DETEC	M024	1	1
119	5541	ROUTINE SEARCH, NEXT	M027	1	1
121	5545	ROUTINE SEARCH, REBA, SECTION	M023	1	1
122	5546	ROUTINE SEARCH	M024	1	1
126	5745	TITLE DIRECT FIRE ROUTINES			
125	5774	ROUTINE STL, CHECK	DF01	1	1
127	5856	ROUTINE CHECK, DEAL	DF02	1	1
128	5916	ROUTINE CHECK, ENGAGEMENT	DF03	1	1
130	5940	ROUTINE CHECK, FORCE	DF04	1	1
131	5997	ROUTINE CHECK, FOR, MINES	DF05	1	1
134	6117	ROUTINE CHECK, LIST	DF06	1	1
135	6137	ROUTINE CHECK, PROX	DF07	1	1
138	6236	ROUTINE CHECK, STREN	DF08	1	1
139	6302	ROUTINE SEARCH, UNIT	DF09	1	1
142	6473	ROUTINE SEARCH, BATTLE	DF10	1	1
143	6516	ROUTINE SEARCH, BATTLE	DF11	1	1
144	6672	ROUTINE SEARCH, COMPUTE	DF12	1	1
149	6866	TITLE INDIRECT FIRE ROUTINES			
150	6881	ROUTINE AC, DETECTION	IF01	1	1
155	7087	ROUTINE ATTRIT, SENSOR	IF02	1	1
159	7286	ROUTINE SEARCH, PH, DEG	IF03	1	1

Figure 3.6  
Configuration Control Form Used for Metrics



```

430 IF TP.SENSOR.TYPE(FM.TGT(FM)) = "RPV" AND
431 TR.PGM.STATUS(FM.TGT(FM)) = TRUE
432 **PERF THE ADJUSTING RPV TO CONTINUE ITS SEARCH
433
434 LET TIME.ALTR.SENSOR.ID(FM.TGT(FM))=REAL.F(FM.TOF.TIME(FM))/
435 3600.
436
437 RESUME REMOTE.PILOT.VEHICLE CALLED TR.SENSOR.ID(FM.TGT(FM))----->(324)
438 ALWAYS
439
440 IF FM IS IN AN FO.CUR.FM.LIST
441 **WHEN THE FIRE MISSION WAS FO ADJUSTED, RESTART THE FO
442 **SO THAT HE MAY CONTINUE HIS SEARCH UNLESS HE HAS
443 **ANOTHER FIRE MISSION TO ADJUST
444
445 REMOVE THE FM FROM FO.CUR.FM.LIST(FO)
446 DESTROY EX.FIRE.MISSION CALLED FM.EX.FIRE.MISSION(FM)
447 STORE 0 IN FM.EX.FIRE.MISSION(FM)
448 IF FO.CUR.FM.LIST(FO) IS EMPTY
449 4 | IF TR.PGM.STATUS(FM.TGT(FM)) = TRUE AND FO IS NOT IN EV.S
450 LET TIME.A(FO) = REAL.F(FM.TOF.TIME(FM))/3600.
451 RESUME FORWARD.OBSERVER CALLED FO----->( 55)
452 ALWAYS
453 3 | ELSE
454 IF TR.PGM.STATUS(FM.TGT(F.FO.CUR.FM.LIST(FO))) HE TRUE AND
455 TR.PGM.STATUS(FM.TGT(FM)) = TRUE AND FO IS NOT IN EV.S
456 RESUME FORWARD.OBSERVER CALLED FO----->( 55)
457 ELSE
458 6 | IF TR.PGM.STATUS(FM.TGT(FM)) = TRUE
459 AND F.FO.CUR.FM.LIST(FO) IS NOT IN EV.S
460 RESUME FIRE.MISSION CALLED F.FO.CUR.FM.LIST(FO)----->( 14)
461 ALWAYS
462 ALWAYS
463
464 **AT THE END OF THE FIRE MISSION, CLEAN UP ALL
465 **SHIP RELATIONSHIPS ASSOCIATED WITH THIS
466 **FIRE MISSION SO THAT THEY ARE NOT LEFT HANGING
467 **AFTER THE PROCESS KILLS ITSELF
468 REMOVE THE FM FROM THE AT.FM.QUEUE(.RTY)
469 DESTROY EX.FIRE.MISSION CALLED FM.EX.FIRE.MISSION(FM)
470 CALL GRY.FM.DF4-----
471 GRY.FM
472
473 REMOVE THE FM FROM THE TP.FM.LIST(FM.TGT(FM))
474 IF TR.FM.LIST(FM.TGT(FM)) IS EMPTY
475 IF FM.TGT(FM) IS NOT IN EV.S
476 8 | DEACTIVATE THE TARGET REPORT CALLED FM.TGT(FM) NON----->(007)
477 ALWAYS
478
479
480
481
482
483
484
485
486
487
488
489

```

Figure 3.7

Level of IF Nesting



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
1	ROUTINE BTRY.EFFECTS	94
2	PROCESS SHOOT.OUT	61
3	PROCESS HELICOPTER.FIRE	48
4	PROCESS TARGET.REPORT	47
5	PROCESS FIRE.MISSION	45
6	ROUTINE FD.DETECTION	43
7	PROCESS HC.ARRIVE.BATTLE	41
8	EVENT AD.ENGAGEMENT	38
9	PROCESS AC.ATK.TGT	37
10	ROUTINE FINAL.COVERAGE	35
11	ROUTINE FA.BN.MOVEMENT	33
12	PROCESS HEL.TARGET.ACQUISITION	32
13	ROUTINE AD.DETECTION	31
14	ROUTINE CHECK.CAS.CONSTRAINTS	31
15	PROCESS ASSESSMENT	30
16	ROUTINE FLIGHT.PATH	30
17	EVENT OFF.LINE.ATTRITION	30
18	ROUTINE FA.BN.ASGN	29
19	PROCESS HC.RETURN.FARRP	29
20	PROCESS AIR.OBSERVER	28
21	EVENT START.BATTLE	28
22	PROCESS CAS.MISSION	27
23	ROUTINE ATTRIT.SENSOR	26
24	ROUTINE PGM.MSN.ASGN	26
25	ROUTINE REQUEST.SMOKE	26
26	ROUTINE UNIT.INPUT	26
27	ROUTINE PK.COMPUTE	25
28	ROUTINE AD.SHOOT	23
29	ROUTINE MINE.EFFECTS	23
30	ROUTINE UNIT.ENVIR	23
31	ROUTINE REQUEST.ILLUM	22
32	ROUTINE AC.BOMB.EFFECTS	20
33	ROUTINE EST.COVERAGE	20
34	ROUTINE ANALYSIS.OUTPUT	19
35	PROCESS ARTY.ASSESS	13
36	ROUTINE CHECK.PROX	18
37	PROCESS FORWARD.OBSERVER	13
38	ROUTINE AC.DF.EFFECTS	17

Figure 3-8  
Modules Ranked by IF Tests



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
39	ROUTINE TACAIR.INPLT	17
40	ROUTINE EMPLOY.HELICOPTERS	16
41	ROUTINE SENSOR.INPUT	16
42	ROUTINE ENC.CAS.MISSION	15
43	EVENT GET.NX.ORD	15
44	ROUTINE BTRY.INPUT	14
45	ROUTINE CHANGE.LOC	14
46	ROUTINE INTER.BATTLE	14
47	PROCESS MINE.ASSESS	14
48	ROUTINE MINE.DELAY	14
49	ROUTINE CFR.DETECTION	13
50	ROUTINE DEAD.UNIT	13
51	ROUTINE HE.OR.ICM.COMPUTATION	13
52	FUNCTION HE.WLA	13
53	EVENT HELO.ENGAGEMENT	13
54	ROUTINE NEW.SEGMENT	13
55	ROUTINE READ.ORDERS	13
56	ROUTINE SMOKE.EFFECTS	13
57	ROUTINE CAS.EVAL	12
58	ROUTINE ILLUM.EFFECTS	12
59	EVENT START.MOVE	12
60	ROUTINE WHAT.NEXT	12
61	ROUTINE AMMO.RPT	11
62	ROUTINE LINE.OF.SIGHT	11
63	ROUTINE PREPARE.LIST	11
64	ROUTINE REQUEST.WD.FASCAM	11
65	ROUTINE RPV.DETECTION	11
66	EVENT UPDATE.LOC	11
67	ROUTINE CHECK.FOR.MINES	10
68	ROUTINE DUST.EFFECTS	10
69	ROUTINE FIND.START.TIME	10
70	ROUTINE GENERAL.BATTLE	10
71	ROUTINE TARGET.ANALYSIS	10
72	ROUTINE BLOCK.LOS	9
73	EVENT BTL.ENDED	9
74	EVENT CFR.OPERATOR	9
75	ROUTINE OUTPUT.ATTRITION	9
76	ROUTINE PIR.DETECTION	9

Figure 3-8

Modules Ranked by IF Tests

Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
77	ROUTINE WEIGHTED.VOLLEYS	9
78	ROUTINE ENQ.FEBA.SET	8
79	FUNCTION FEBA.BAND	8
80	EVENT FEBA.SORTIE	5
81	ROUTINE FILE.FD.SCHD	9
82	ROUTINE HC.EMPTY	3
83	ROUTINE KV.PRINT	4
84	ROUTINE ORIENTATION	3
85	ROUTINE OUTPUT.EXPENDITURES	8
86	ROUTINE PDB.DETECTION	8
87	PROCESS REMOTE.PILOT.VEHICLE	3
88	ROUTINE SIZE.ESTIMATE	3
89	ROUTINE ADJUST	7
90	PROCESS AIRBORNE.RADAR	7
91	ROUTINE LOCATE.SEARCH.AREA	7
92	ROUTINE MARGINAL.EFFECTS.ADJ	7
93	ROUTINE REQUEST.FASCAM	7
94	ROUTINE SEARCH.COVERAGE	7
95	ROUTINE STL.CHECK	6
96	EVENT CFR.ON	5
97	ROUTINE DESTROY.ORD	5
98	ROUTINE FEBA.INITIAL	6
99	ROUTINE FIN.BATTLE	6
100	ROUTINE HELLA.INPUT	5
101	ROUTINE LOS.CHECK	5
102	ROUTINE MINE.INPUT	5
103	ROUTINE NOISE.DEGRADE	6
104	ROUTINE REQUEST.DEF.FASCAM	5
105	PROCESS WITH.DRAW	5
106	ROUTINE ANGLE.COMPUTE	5
107	EVENT CFR.ACTIVATION	5
108	ROUTINE CHECK.DEAD	5
109	ROUTINE CHECK.FORCE	5
110	ROUTINE COMPARE.TRS	5
111	ROUTINE CREATE.FORCE	5
112	ROUTINE END.MOVE	5
113	EVENT ENGAGEMENT	5
114	ROUTINE HC.COMPUTE.TIMES	5

Figure 3-8  
Modules Ranked by IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
115	ROUTINE HEL.RANGE.COMPUTE	5
116	FUNCTION ICM.WLA	5
117	ROUTINE LOCATE.SECTOR	5
118	EVENT PDB.OPERATOR	5
119	PROCESS PHOTO.IP.FLIGHT	5
120	ROUTINE PLAT.COUNT	5
121	ROUTINE PROX.CHECK	5
122	ROUTINE SWITCH.FO	5
123	ROUTINE UNIT.PRIORITY	5
124	ROUTINE VOLLEY	5
125	EVENT ACT.ATK	4
126	ROUTINE AR.DETECTION	4
127	ROUTINE CBR.DEGRADE	4
128	ROUTINE CHECK.LIST	4
129	FUNCTION COMBINATIONS	4
130	ROUTINE CONTRAST.TO.FREQ	4
131	ROUTINE EMPTY	4
132	ROUTINE EQ.TE.INPUT	4
133	ROUTINE FARRP.CHECK	4
134	ROUTINE FARRP.INPUT	4
135	ROUTINE GAMMA.F	4
136	ROUTINE RESET.FEBA.SECTOR	4
137	ROUTINE SMOKE.COMPUTATION	4
138	EVENT START.ARTY.MOVEMENT	4
139	EVENT ACT.MOVDIS	3
140	EVENT ACT.REINF	3
141	FUNCTION AR.PROB.DETECT	3
142	ROUTINE CAT.TU.INPUT	3
143	EVENT CHANGE.LITE	3
144	ROUTINE CHK.FD.TR	3
145	FUNCTION COLLISION	3
146	ROUTINE DECIDE	3
147	ROUTINE DEQ.FEBA.SET	3
148	ROUTINE FBN.FD.INPUT	3
149	ROUTINE FINISH.COMPUTATION	3
150	ROUTINE FORM.TF.LIST	3
151	ROUTINE GET.TERRAIN	3
152	EVENT HC.DEPART.BATTLE	3

Figure 3-8  
Modules Ranked by IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
153	PROCESS HOW.REPAIR	3
154	ROUTINE ILLUM.COMPUTATION	3
155	ROUTINE INTER.HELO	3
156	ROUTINE KV.SCOREBOARD	3
157	ROUTINE MAIN2	3
158	ROUTINE MIN.MOVE	3
159	EVENT PDB.ACTIVATION	3
160	ROUTINE PRED.POS	3
161	ROUTINE PREP.WITHDRAW	3
162	ROUTINE REIN.ARRIVE	3
163	ROUTINE REM.EFFECTS.COMPUTATION	3
164	ROUTINE REPLACE.HC	3
165	ROUTINE SNAP.R	3
166	ROUTINE TACAIR.DATA.REPORT	3
167	ROUTINE TERM.CHECK	3
168	ROUTINE TIME.TO.DETECT	3
169	ROUTINE AC.MUNS.INPUT	2
170	EVENT ARTY.OCCUPATION	2
171	FUNCTION BTRY.AVAILABLE	2
172	ROUTINE BTRY.FM.DEN	2
173	ROUTINE BTRY.FM.ENG	2
174	ROUTINE CHK.COMP.TR	2
175	ROUTINE COMPUTE.WD	2
176	ROUTINE EST.MIL.WORTH	2
177	ROUTINE FASCAM.COMPUTATION	2
178	ROUTINE FD.EFFECTS.REQ	2
179	ROUTINE FILE.KAD.SENSOR	2
180	EVENT INIT.PREPLAN.CAS	2
181	ROUTINE MPT.TO.FREQ	2
182	ROUTINE NORMAL.F	2
183	ROUTINE PROB.INF	2
184	ROUTINE SEARCH	2
185	EVENT SET.DEBUG	2
186	ROUTINE SMOKE.INPUT	2
187	EVENT STOP.ARTY.MOVEMENT	2
188	ROUTINE TB.INPUT	2
189	ROUTINE TEMPERATURE.ATTENUATION	2
190	ROUTINE BETWEEN.ROUTINE	1

Figure 3-8  
Modules Ranked by IF Tests  
Continued



RANK	MODULE NAME	IF TESTS
191	EVENT CFR.OFF	1
192	ROUTINE CHECK.ENGAGEMENT	1
193	ROUTINE COMBINE.TRS	1
194	ROUTINE DQ.CMSN.QUEUE	1
195	EVENT DQ.OLD.SORTIE.QUEUE	1
196	FUNCTION EST.RANGE	1
197	FUNCTION EST.TR.RANGE	1
198	ROUTINE EXPONENTIAL.F	1
199	ROUTINE FDC.TR.DEN	1
200	ROUTINE FDC.TR.ENQ	1
201	ROUTINE HC.DISENGAGE	1
202	ROUTINE ILLUM.INPUT	1
203	ROUTINE INITIAL.MOVE	1
204	ROUTINE LINE.CIRCLE	1
205	ROUTINE MPOB.INPUT	1
206	ROUTINE ORD.MOVCOP	1
207	ROUTINE PGM.INPUT	1
208	ROUTINE POSITION.OUT	1
209	ROUTINE PROB.TIME	1
210	ROUTINE PROX.POS	1
211	ROUTINE SEGMENT.ADJUST	1
212	EVENT SEND.TEAM	1
213	ROUTINE SYS.INPUT	1
214	ROUTINE TF.INPUT	1
215	ROUTINE UNIT.ASSIGNMENT	1
216	ROUTINE VIS.INPUT	1
217	EVENT ACT.DEF	0
218	EVENT ACT.MOVCOR	0
219	FUNCTION ACT.RANGE	0
220	EVENT CHANGE.WEATHER	0
221	ROUTINE CHECK.STREN	0
222	ROUTINE COMPUTE.D	0
223	ROUTINE COPY	0
224	ROUTINE CREATE.TEAMS	0
225	ROUTINE DECISION.INPUT	0
226	EVENT END.SIMULATION	0
227	ROUTINE ERROR.STOP	0
228	ROUTINE FRAC.COMPUTE	0

Figure 3-8  
Modules Ranked by IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	IF TESTS
229	ROUTINE HEADING	0
230	ROUTINE INIT.REINF	0
231	ROUTINE INITIAL.DETECT	0
232	ROUTINE JOHNSON.CRITERIA	0
233	ROUTINE KV.INPUT	0
234	ROUTINE MADS.INPUT	0
235	PROGRAM MAIN	0
236	ROUTINE MAIN1	0
237	ROUTINE MAIN3	0
238	ROUTINE MAO.INPUT	0
239	ROUTINE MCFR.INPUT	0
240	ROUTINE MFC.INPUT	0
241	EVENT MOVE	0
242	ROUTINE MUNS.INPUT	0
243	PROGRAM OLDER.VERSION.PREAMBLE	0
244	ROUTINE OPEN.INPUT.OUTPUT.FILES	0
245	ROUTINE ORD.ATK	0
246	ROUTINE ORD.DEF	0
247	ROUTINE ORD.MOVDIS	0
248	ROUTINE ORD.REINF	0
249	ROUTINE P.E.M.INPUT	0
250	ROUTINE PK.INPUT	0
251	ROUTINE POSITION	0
252	EVENT POSITION.REPORT	0
253	PROGRAM PREAMBLE	0
254	ROUTINE PROXIMITY.REQ	0
255	ROUTINE RANGE.COMPUTE	0
256	ROUTINE RJL.EN.INPUT	0
257	EVENT SCHEDULE.ARTY.MOVEMENT	0
258	ROUTINE SNAP2	0
259	ROUTINE ST.INPUT	0
260	FUNCTION STAY.TIME	0
261	ROUTINE SUBM.INPUT	0
262	ROUTINE T3F.INPUT	0
263	ROUTINE TIME.REQ	0
264	ROUTINE TT.FACTORS.INPUT	0
265	ROUTINE TYPE.WEAPON.INPUT	0
266	ROUTINE WEISULL.F	0

Figure 3-8  
Modules Ranked by IF Tests  
Continued



second report lists the modules ranked by functional IF tests, that is, the total number of IFs minus the number of IFs controlling debug output (see Figure 3.9). The third report lists the modules by maximum IF nesting depth (see Figure 3.10).

The information contained in these reports was then analyzed. Since the IF tests for debugging are not frequently utilized during COSAGE production runs, it was decided to use the number of functional IFs (adjusted by subtracting debug IFs) as the cyclomatic number for the control complexity metric. As mentioned previously, a routine that has a cyclomatic number greater than 10 (i.e., more than 10 independent paths) will probably not be fully tested. Further, if a routine is not fully tested, then its reliability and maintainability are suspect.

Referring to Figure 3.9, 61 modules in COSAGE have cyclomatic numbers greater than 10. This represents 23% of the COSAGE modules. Of these 61, 41 are routines (representing approximately 21% of all COSAGE routines), 14 are processes (representing approximately 74% of all COSAGE processes), 5 are events (representing approximately 14% of all COSAGE events), and 1 is a function (representing approximately 9% of all functions).

The obvious category which warrants further investigation are processes, since approximately 74% of all the COSAGE processes have cyclomatic numbers (control complexity metrics) greater than 10. Recommended changes to reduce the number of independent paths in these processes are discussed in Section 4.8 of the report.

There are 33 modules (including 8 processes) that contain a maximum depth of IF nesting greater than 3 (see Figure 3.10). The processes are particularly troublesome because they

RANK	MODULE NAME	FUNCTIONAL IF TESTS
1	ROUTINE BTRY.EFFECTS	88
2	PROCESS SHOOT.OUT	61
3	PROCESS FIRE.MISSION	40
4	PROCESS HELICOPTER.FIRE	38
5	PROCESS TARGET.REPORT	36
6	ROUTINE FO.DETECTION	35
7	PROCESS HC.ARRIVE.BATTLE	34
8	ROUTINE FINAL.COVERAGE	32
9	PROCESS ASSESSMENT	30
10	EVENT OFF.LINE.ATTRITION	30
11	ROUTINE FA.BN.MOVEMENT	29
12	PROCESS AC.ATK.TGT	27
13	EVENT AD.ENGAGEMENT	27
14	PROCESS AIR.OBSERVER	27
15	ROUTINE FLIGHT.PATH	26
16	ROUTINE UNIT.INPUT	26
17	PROCESS HC.RETURN.FARPP	25
18	ROUTINE PK.COMPUTE	25
19	EVENT START.BATTLE	25
20	ROUTINE AD.DETECTION	24
21	ROUTINE CHECK.CAS.CONSTRAINTS	24
22	PROCESS HEL.TARGET.ACQUISITION	24
23	ROUTINE REQUEST.SMOKE	24
24	ROUTINE FA.BN.ASGN	22
25	ROUTINE UNIT.ENVIR	21
26	ROUTINE AC.BOMB.EFFECTS	20
27	ROUTINE MINE.EFFECTS	20
28	ROUTINE PGM.MSN.ASGN	20
29	ROUTINE REQUEST.ILLUM	20
30	ROUTINE AD.SHOOT	19
31	ROUTINE ANALYSIS.OUTPUT	19
32	PROCESS CAS.MISSION	19
33	PROCESS ARTY.ASSESS	18
34	ROUTINE ATTRIT.SENSOR	18
35	ROUTINE CHECK.PROX	18
36	ROUTINE TACAIR.INPUT	17
37	ROUTINE AC.OF.EFFECTS	16
38	ROUTINE SENSOR.INPUT	16

Figure 3-9  
Modules Ranked by Functional IF Tests

# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
39	ROUTINE END.CAS.MISSION	13
40	EVENT GET.NX.ORD	13
41	ROUTINE STRY.INPUT	14
42	ROUTINE CHANGE.LOC	14
43	ROUTINE EST.COVERAGE	14
44	ROUTINE INTER.BATTLE	14
45	ROUTINE CFR.DETECTION	13
46	ROUTINE DEAD.UNIT	13
47	ROUTINE MINE.DELAY	13
48	ROUTINE READ.ORDERS	13
49	ROUTINE HE.LO.R.ICM.COMPUTATION	12
50	FUNCTION HE.WLA	12
51	ROUTINE WHAT.NEXT	12
52	ROUTINE AMMO.RPT	11
53	ROUTINE CAS.EVAL	11
54	ROUTINE EMPLOY.HELICOPTERS	11
55	PROCESS FORWARD.OBSERVER	11
56	ROUTINE LINE.OF.SIGHT	11
57	PROCESS MINE.ASSESS	11
58	ROUTINE NEW.SEGMENT	11
59	ROUTINE REQUEST.WO.FASCAM	11
60	ROUTINE RPV.DETECTION	11
61	EVENT START.MOVE	11
62	ROUTINE CHECK.FOR.MINES	10
63	ROUTINE DUST.EFFECTS	10
64	ROUTINE FIND.START.TIME	10
65	ROUTINE GENERAL.BATTLE	10
66	ROUTINE PREPARE.LIST	10
67	ROUTINE BLOCK.LOS	9
68	EVENT CFR.OPERATOR	9
69	ROUTINE ILLUM.EFFECTS	9
70	ROUTINE OUTPUT.ATTRITION	9
71	ROUTINE PIR.DETECTION	9
72	ROUTINE SMOKE.EFFECTS	9
73	EVENT UPDATE.LOC	8
74	EVENT FEBA.SORTIE	8
75	ROUTINE FILE.FD.SCHD	8
76	EVENT HELD.ENGAGEMENT	8

Figure 3-9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
77	ROUTINE KV.PRINT	3
78	ROUTINE ORIENTATION	3
79	ROUTINE OUTPUT.EXPENDITURES	3
80	ROUTINE PDB.DETECTION	3
81	PROCESS REMOTE.PILOT.VEHICLE	3
82	ROUTINE SIZE.ESTIMATE	3
83	ROUTINE TARGET.ANALYSIS	3
84	ROUTINE ADJUST	7
85	PROCESS AIRBORNE.PADAR	7
86	EVENT BTL.ENDED	7
87	ROUTINE ENQ.FEBA.SET	7
88	FUNCTION FEBA.BAND	7
89	ROUTINE MARGINAL.EFFECTS.ADJ	7
90	ROUTINE SEARCH.COVERAGE	7
91	EVENT CFR.ON	6
92	ROUTINE DESTROY.ORD	6
93	ROUTINE FIN.BATTLE	6
94	ROUTINE HE.LA.INPUT	6
95	ROUTINE LOCATE.SEARCH.AREA	6
96	ROUTINE LOS.CHECK	6
97	ROUTINE MINE.INPUT	6
98	ROUTINE NOISE.DEGRADE	6
99	ROUTINE REQUEST.DEF.FASCAM	6
100	ROUTINE REQUEST.FASCAM	6
101	ROUTINE WEIGHTED.VOLLEYS	6
102	PROCESS WITH.DRAW	6
103	EVENT CFR.ACTIVATION	5
104	ROUTINE CHECK.DEAD	5
105	ROUTINE CHECK.FORCE	5
106	ROUTINE COMPARE.TRS	5
107	ROUTINE CREATE.FORCE	5
108	ROUTINE END.MOVE	5
109	ROUTINE FEBA.INITIAL	5
110	FUNCTION ICM.WLA	5
111	EVENT PDB.OPERATOR	5
112	ROUTINE PLAT.COUNT	5
113	ROUTINE PROX.CHECK	5
114	ROUTINE SWITCH.FO	5

Figure 3-9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
115	ROUTINE VOLLEY	5
116	EVENT ACT. ATK	4
117	ROUTINE ANGLE.COMPUTE	4
118	ROUTINE STL.CHECK	4
119	ROUTINE CFR.DEGRADE	4
120	FUNCTION COMBINATIONS	4
121	ROUTINE CONTRAST.TO.FREQ	4
122	ROUTINE EMPTY	4
123	ROUTINE EQ.TE.INPUT	4
124	ROUTINE FARRP.INPUT	4
125	ROUTINE GAMMA.F	4
126	ROUTINE HC.COMPUTE.TIMES	4
127	ROUTINE HEL.RANGE.COMPUTE	4
128	PROCESS PHOTO.IR.FLIGHT	4
129	EVENT ACT.MOVDIS	3
130	EVENT ACT.REINF	3
131	ROUTINE AR.DETECTION	3
132	FUNCTION AR.PROB.DETECT	3
133	EVENT CHANGE.LITE	3
134	ROUTINE CHK.FD.TP	3
135	FUNCTION COLLISION	3
136	ROUTINE DECIDE	3
137	EVENT ENGAGEMENT	3
138	ROUTINE FARRP.CHECK	3
139	ROUTINE FBW.FD.INPUT	3
140	ROUTINE GET.TERRAIN	3
141	PROCESS HDW.REPAIR	3
142	ROUTINE KV.SCOREBOARD	3
143	ROUTINE MIN.MOVE	3
144	EVENT PDS.ACTIVATION	3
145	ROUTINE PRED.POS	3
146	ROUTINE PREP.WITHDRAW	3
147	ROUTINE REIN.ARRIVE	3
148	ROUTINE RESET.FEBA.SECTOR	3
149	ROUTINE SMOKE.COMPUTATION	3
150	ROUTINE SNAP.R	3
151	EVENT START.ARTY.MOVEMENT	3
152	ROUTINE TACAIR.DATA.REPORT	3

Figure 3.9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
153	ROUTINE TERM.CHECK	3
154	ROUTINE UNIT.PRIORITY	3
155	EVENT ARTY.OCCUPATION	2
156	ROUTINE BTRY.FM.DEG	2
157	ROUTINE BTRY.FM.ENG	2
158	ROUTINE CAT.TU.INPUT	2
159	ROUTINE CHECK.LIST	2
160	ROUTINE CHK.COMP.TR	2
161	ROUTINE COMPUTE.WD	2
162	ROUTINE DEG.FEBA.SET	2
163	ROUTINE FD.EFFECTS.REQ	2
164	ROUTINE FINISH.COMPUTATION	2
165	ROUTINE FORM.TF.LIST	2
166	EVENT HC.DEPART.BATTLE	2
167	ROUTINE HC.EMPTY	2
168	ROUTINE ILLUM.COMPUTATION	2
169	ROUTINE INTER.HELQ	2
170	ROUTINE LOCATE.SECTOR	2
171	ROUTINE MAIN2	2
172	ROUTINE MRT.TO.FREQ	2
173	ROUTINE NORMAL.F	2
174	ROUTINE PROB.INF	2
175	ROUTINE SEARCH	2
176	EVENT SET.DEBUG	2
177	ROUTINE SMOKE.INPUT	2
178	EVENT STOP.ARTY.MOVEMENT	2
179	ROUTINE TB.INPUT	2
180	ROUTINE TEMPERATURE.ATTENUATION	2
181	ROUTINE TIME.TO.DETECT	2
182	ROUTINE AC.MUNS.INPUT	1
183	FUNCTION STRY.AVAILABLE	1
184	EVENT CFR.OFF	1
185	ROUTINE CHECK.ENGAGEMENT	1
186	ROUTINE COMBINE.TRS	1
187	ROUTINE EST.MIL.WORTH	1
188	ROUTINE EXPONENTIAL.F	1
189	ROUTINE FASCAM.COMPUTATION	1
190	ROUTINE FDC.TR.DEG	1

Figure 3-9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
191	ROUTINE FOC.TR.ENG	1
192	ROUTINE FILE.KAD.SENSOR	1
193	ROUTINE ILLUM.INPUT	1
194	EVENT INIT.PREPLAN.CAS	1
195	ROUTINE INITIAL.MOVE	1
196	ROUTINE LINE.CIRCLE	1
197	ROUTINE MPDB.INPUT	1
198	ROUTINE CRD.MOVCUR	1
199	ROUTINE PSM.INPUT	1
200	ROUTINE POSITION.OUT	1
201	ROUTINE PROB.TIME	1
202	ROUTINE PROX.POS	1
203	ROUTINE REM.EFFECTS.COMPUTATION	1
204	ROUTINE REPLACE.HC	1
205	ROUTINE SEGMENT.ADJUST	1
206	ROUTINE SYS.INPUT	1
207	ROUTINE TR.INPUT	1
208	ROUTINE VIS.INPUT	1
209	EVENT ACT.DEF	0
210	EVENT ACT.MOVCUR	0
211	FUNCTION ACT.RANGE	0
212	ROUTINE BETWEEN.ROUTINE	0
213	EVENT CHANGE.WEATHER	0
214	ROUTINE CHECK.STREN	0
215	ROUTINE COMPUTE.C	0
216	ROUTINE COPY	0
217	ROUTINE CREATE.TEAMS	0
218	ROUTINE DECISION.INPUT	0
219	ROUTINE DL.CMSN.QUEUE	0
220	EVENT DL.OLD.SORTIE.QUEUE	0
221	EVENT END.SIMULATION	0
222	ROUTINE ERROR.STOP	0
223	FUNCTION EST.RANGE	0
224	FUNCTION EST.TR.RANGE	0
225	ROUTINE FRAC.COMPUTE	0
226	ROUTINE HC.DISENGAGE	0
227	ROUTINE HEADING	0
228	ROUTINE INIT.REINF	0

Figure 3-9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	FUNCTIONAL IF TESTS
229	ROUTINE INITIAL.DETECT	0
230	ROUTINE JOHNSON.CRITERIA	0
231	ROUTINE KV.INPUT	0
232	ROUTINE MAOS.INPUT	0
233	PROGRAM MAIN	0
234	ROUTINE MAIN1	0
235	ROUTINE MAIN3	0
236	ROUTINE MAO.INPUT	0
237	ROUTINE MCFR.INPUT	0
238	ROUTINE MFO.INPUT	0
239	EVENT MOVE	0
240	ROUTINE MUNS.INPUT	0
241	PROGRAM OLCEP.VERSION.PREAMBLE	0
242	ROUTINE OPEN.INPUT.OUTPUT.FILES	0
243	ROUTINE ORD.ATK	0
244	ROUTINE ORD.OFF	0
245	ROUTINE ORD.MOVDIS	0
246	ROUTINE ORD.REIVE	0
247	ROUTINE P.E.M.INPUT	0
248	ROUTINE PK.INPUT	0
249	ROUTINE POSITION	0
250	EVENT POSITION.REPORT	0
251	PROGRAM PREAMBLE	0
252	ROUTINE PROXIMITY.REQ	0
253	ROUTINE RANGE.COMPUTE	0
254	ROUTINE RJL.EN.INPUT	0
255	EVENT SCHEDULE.APTY.MOVEMENT	0
256	EVENT SEND.TEAM	0
257	ROUTINE SNAPZ	0
258	ROUTINE ST.INPUT	0
259	FUNCTION STAY.TIME	0
260	ROUTINE SUB4.INPUT	0
261	ROUTINE TBF.INPUT	0
262	ROUTINE TIME.REQ	0
263	ROUTINE TT.FACTORS.INPUT	0
264	ROUTINE TYPE.WEAPON.INPUT	0
265	ROUTINE UNIT.ASSIGNMENT	0
266	ROUTINE WEIBULL.F	0

Figure 3-9  
Modules Ranked by Functional IF Tests  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
1	EVENT GET.NX.DRD	7
2	ROUTINE CHECK.CAS.CONSTRAINTS	6
3	ROUTINE DESTROY.ORD	6
4	PROCESS FIRE.MISSION	6
5	ROUTINE PK.COMPUTE	6
6	PROCESS TARGET.REPORT	6
7	ROUTINE WHAT.NEXT	6
8	ROUTINE AC.DF.EFFECTS	5
9	ROUTINE GTRY.EFFECTS	5
10	ROUTINE CHECK.PROX	5
11	ROUTINE MINE.EFFECTS	5
12	ROUTINE POB.DETECTION	5
13	ROUTINE PLAT.COUNT	5
14	ROUTINE READ.OFDEPS	5
15	ROUTINE UNIT.FNVP	5
16	ROUTINE AC.BOMB.EFFECTS	4
17	EVENT AD.ENGAGEMENT	4
18	PROCESS APTY.ASSESS	4
19	PROCESS CAS.MISSION	4
20	ROUTINE CHANGE.LOC	4
21	ROUTINE DEAD.UNIT	4
22	ROUTINE FA.BN.ASGN	4
23	ROUTINE FA.BN.MOVEMENT	4
24	ROUTINE FLIGHT.PATH	4
25	PROCESS HC.RETURN.FAKRP	4
26	PROCESS HEL.TARGET.ACQUISITION	4
27	PROCESS HELICOPTER.FIRE	4
28	ROUTINE MINE.DELAY	4
29	ROUTINE PREPARE.LIST	4
30	ROUTINE REQUEST.SMOKE	4
31	ROUTINE REQUEST.WD.FASCAM	4
32	PROCESS SHOOT.OUT	4
33	EVENT START.BATTLE	4
34	EVENT ACT.REINF	3
35	ROUTINE AD.SHOOT	3
36	PROCESS AIR.OBSERVER	3
37	FUNCTION AP.PROS.DETECT	3
38	PROCESS ASSESSMENT	3

Figure 3-10  
Modules Ranked by Maximum IF Depth



# SCIENCE APPLICATIONS, INC.

RANK		MODULE NAME	MAXIMUM DEPTH
39	ROUTINE	ATTRIT. SENSOR	3
40	EVENT	STL. ENDED	3
41	ROUTINE	STRY. INPUT	3
42	ROUTINE	CAS. EVAL	3
43	EVENT	CRK. ACTIVATION	3
44	ROUTINE	CRK. DETECTION	3
45	EVENT	CRK. OPERATOR	3
46	ROUTINE	CHK. FC. TR	3
47	ROUTINE	COMPARE. TPS	3
48	ROUTINE	CONTRAST. TO. PPAL	3
49	ROUTINE	EMPLOY. HELICOPTERS	3
50	ROUTINE	END. CAS. MISSION	3
51	ROUTINE	END. MOVE	3
52	ROUTINE	ENV. FC. SET	3
53	ROUTINE	EST. COVERAGE	3
54	ROUTINE	FILE. FC. SCHO	3
55	ROUTINE	FINAL. COV-RAGE	3
56	ROUTINE	FC. DETECTION	3
57	PROCESS	FORWARD. OBSERVED	3
58	ROUTINE	GET. TERRAIN	3
59	PROCESS	HO. ARRIVE. BATTLE	3
60	ROUTINE	HE. OR. ICM. COMPUTATION	3
61	EVENT	HELO. ENGAGEMENT	3
62	ROUTINE	LOS. CHECK	3
63	PROCESS	MINE. ACCESS	3
64	ROUTINE	NEW. SEGMENT	3
65	EVENT	OFF. LINE. ATTRITION	3
66	ROUTINE	PIP. DETECTION	3
67	ROUTINE	REIN. ARRIVE	3
68	ROUTINE	REQUEST. PASCAM	3
69	ROUTINE	REQUEST. ILLUM	3
70	ROUTINE	RPV. DETECTION	3
71	ROUTINE	SEARCH. COVERAGE	3
72	ROUTINE	SNAP. R	3
73	EVENT	START. MOVE	3
74	ROUTINE	SWITCH. FC	3
75	ROUTINE	TACAIR. INPUT	3
76	ROUTINE	UNIT. INPUT	3

Figure 3-10  
Modules Ranked by Maximum IF Depth  
Continued

# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
77	EVENT UPDATE.LOC	3
78	PROCESS AC.ATK.TGT	2
79	EVENT ACT.ATK	2
80	EVENT ACT.MOVDIS	2
81	ROUTINE ADJUST	2
82	ROUTINE AMMO.RPT	2
83	ROUTINE ANALYSIS.OUTPUT	2
84	ROUTINE ANGLE.COMPUTE	2
85	ROUTINE AC.DETECTION	2
86	ROUTINE BLOCK.LOS	2
87	ROUTINE BTL.CHECK	2
88	ROUTINE BTRY.FM.DEG	2
89	ROUTINE CAT.TU.INPUT	2
90	ROUTINE CBR.DEGRADE	2
91	EVENT CHANGE.LITE	2
92	ROUTINE CHECK.FOR.MINES	2
93	ROUTINE CHECK.FORCE	2
94	ROUTINE CHK.COMP.TR	2
95	ROUTINE CREATE.FORCE	2
96	ROUTINE DUST.EFFECTS	2
97	ROUTINE EMPTY	2
98	ROUTINE EQ.TE.INPUT	2
99	ROUTINE EST.MIL.WORTH	2
100	ROUTINE FARRP.CHECK	2
101	ROUTINE FB%FD.INPUT	2
102	FUNCTION FEBA.BAND	2
103	EVENT FEBA.SOFTIE	2
104	ROUTINE FINISH.COMPUTATION	2
105	ROUTINE FORM.TF.LIST	2
106	ROUTINE GENERAL.BATTLE	2
107	ROUTINE HE.LA.INPUT	2
108	FUNCTION HE.WLA	2
109	ROUTINE HEL.RANGE.COMPUTE	2
110	PROCESS HOW.REPAIR	2
111	ROUTINE ILLUM.COMPUTATION	2
112	ROUTINE ILLUM.EFFECTS	2
113	ROUTINE INTER.BATTLE	2
114	ROUTINE KV.PRINT	2

Figure 3-10  
Modules Ranked by Maximum IF Depth  
Continued

# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
115	ROUTINE KV.SCOREBOARD	2
116	ROUTINE LINE.OF.SIGHT	2
117	ROUTINE LOCATE.SEARCH.AREA	2
118	ROUTINE MARGINAL.EFFECTS.ADJ	2
119	ROUTINE MINE.INPUT	2
120	ROUTINE NOISE.DEGRADE	2
121	ROUTINE ORIENTATION	2
122	ROUTINE OUTPUT.ATTRITION	2
123	EVENT PDB.OPERATOR	2
124	ROUTINE PGM.MSN.ASGN	2
125	ROUTINE PREP.WITHDRAW	2
126	ROUTINE PROG.INF	2
127	ROUTINE PROX.CHECK	2
128	ROUTINE REPLACE.HC	2
129	ROUTINE REQUEST.DEF.FASCAM	2
130	ROUTINE SENSOR.INPUT	2
131	EVENT SET.DEBUG	2
132	ROUTINE SMOKE.COMPUTATION	2
133	ROUTINE SMOKE.EFFECTS	2
134	ROUTINE TEMPERATURE.ATTENUATION	2
135	ROUTINE UNIT.PRIORITY	2
136	ROUTINE VOLLEY	2
137	ROUTINE WEIGHTED.VOLLEYS	2
138	PROCESS WITH.DRAW	2
139	ROUTINE AC.MUNS.INPUT	1
140	PROCESS AIRBORNE.RADAR	1
141	ROUTINE AR.DETECTION	1
142	EVENT ARTY.OCCUPATION	1
143	ROUTINE BETWEEN.ROUTINE	1
144	FUNCTION BTRY.AVAILABLE	1
145	ROUTINE BTRY.FM.ENG	1
146	EVENT CFR.OFF	1
147	EVENT CFR.ON	1
148	ROUTINE CHECK.DEAD	1
149	ROUTINE CHECK.ENGAGEMENT	1
150	ROUTINE CHECK.LIST	1
151	FUNCTION COLLISION	1
152	FUNCTION COMBINATIONS	1

Figure 3-10  
Modules Ranked by Maximum IF Depth  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
229	ROUTINE HEADING	0
230	ROUTINE INIT.REINF	0
231	ROUTINE INITIAL.DETECT	0
232	ROUTINE JOHNSON.CRITERIA	0
233	ROUTINE KV.INPUT	0
234	ROUTINE MAD5.INPUT	0
235	PROGRAM MAIN	0
236	ROUTINE MAIN1	0
237	ROUTINE MAIN3	0
238	ROUTINE MAO.INPUT	0
239	ROUTINE MCFR.INPUT	0
240	ROUTINE MFO.INPUT	0
241	EVENT MOVE	0
242	ROUTINE MUNS.INPUT	0
243	PROGRAM OLDER.VERSION.PREAMBLE	0
244	ROUTINE OPEN.INPUT.OUTPUT.FILES	0
245	ROUTINE ORD.ATK	0
246	ROUTINE ORD.DEF	0
247	ROUTINE ORD.MOVDIS	0
248	ROUTINE ORD.REINF	0
249	ROUTINE P.E.M.INPUT	0
250	ROUTINE PY.INPUT	0
251	ROUTINE POSITION	0
252	EVENT POSITION.REPORT	0
253	PROGRAM PREAMBLE	0
254	ROUTINE PROXIMITY.REQ	0
255	ROUTINE RANGE.COMPUTE	0
256	ROUTINE RUL.EN.INPUT	0
257	EVENT SCHEDULE.ARTY.MOVEMENT	0
258	ROUTINE SNAP2	0
259	ROUTINE ST.INPUT	0
260	FUNCTION STAY.TIME	0
261	ROUTINE SUBM.INPUT	0
262	ROUTINE T3F.INPUT	0
263	ROUTINE TIME.REQ	0
264	ROUTINE TT.FACTORS.INPUT	0
265	ROUTINE TYPE.WEAPON.INPUT	0
266	ROUTINE WEIBULL.F	0

Figure 3-10  
Modules Ranked by Maximum IF Depth  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
191	EVENT PDB.ACTIVATION	1
192	ROUTINE PGM.INPUT	1
193	PROCESS PHOTO.IR.FLIGHT	1
194	ROUTINE POSITION.OUT	1
195	ROUTINE PRED.POS	1
196	ROUTINE PROB.TIME	1
197	ROUTINE PROX.POS	1
198	ROUTINE REM.EFFECTS.COMPUTATION	1
199	PROCESS REMOTE.PILOT.VEHICLE	1
200	ROUTINE RESET.FEBA.SECTOR	1
201	ROUTINE SEARCH	1
202	ROUTINE SEGMENT.ADJUST	1
203	EVENT SEND.TEAM	1
204	ROUTINE SIZE.ESTIMATE	1
205	ROUTINE SMOKE.INPUT	1
206	EVENT START.ARTY.MOVEMENT	1
207	EVENT STOP.ARTY.MOVEMENT	1
208	ROUTINE SYS.INPUT	1
209	ROUTINE TACAIP.DATA.REPORT	1
210	ROUTINE TARGET.ANALYSIS	1
211	ROUTINE T3.INPUT	1
212	ROUTINE TERM.CHECK	1
213	ROUTINE TIME.TO.DETECT	1
214	ROUTINE TR.INPUT	1
215	ROUTINE UNIT.ASSIGNMENT	1
216	ROUTINE VIS.INPUT	1
217	EVENT ACT.DEF	0
218	EVENT ACT.MOVCOR	0
219	FUNCTION ACT.RANGE	0
220	EVENT CHANGE.WEATHER	0
221	ROUTINE CHECK.STREN	0
222	ROUTINE COMPUTE.O	0
223	ROUTINE COPY	0
224	ROUTINE CREATE.TEAMS	0
225	ROUTINE DECISION.INPUT	0
226	EVENT END.SIMULATION	0
227	ROUTINE ERROR.STOP	0
228	ROUTINE FRAC.COMPUTE	0

Figure 3-10  
Modules Ranked by Maximum IF Depth  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	MAXIMUM DEPTH
153	ROUTINE COMBINE.TRS	1
154	ROUTINE COMPUTE.WD	1
155	ROUTINE DECIDE	1
156	ROUTINE DEQ.FE9A.SET	1
157	ROUTINE DQ.CMSN.QUEUE	1
158	EVENT DQ.OLD.SORTIE.QUEUE	1
159	EVENT ENGAGEMENT	1
160	FUNCTION EST.RANGE	1
161	FUNCTION EST.TR.RANGE	1
162	ROUTINE EXPONENTIAL.F	1
163	ROUTINE FARRP.INPUT	1
164	ROUTINE FASCAM.COMPUTATION	1
165	ROUTINE FD.EFFECTS.REQ	1
166	ROUTINE FDC.TR.DEQ	1
167	ROUTINE FDC.TR.ENQ	1
168	ROUTINE FE9A.INITIAL	1
169	ROUTINE FILE.KAD.SENSOR	1
170	ROUTINE FIN.BATTLE	1
171	ROUTINE FIND.START.TIME	1
172	ROUTINE GAMMA.F	1
173	ROUTINE HC.COMPUTE.TIMES	1
174	EVENT HC.DEPART.BATTLE	1
175	ROUTINE HC.DISENGAGE	1
176	ROUTINE HC.EMPTY	1
177	FUNCTION ICM.WLA	1
178	ROUTINE ILLUM.INPUT	1
179	EVENT INIT.PREPLAN.CAS	1
180	ROUTINE INITIAL.MOVE	1
181	ROUTINE INTER.HELO	1
182	ROUTINE LINE.CIRCLE	1
183	ROUTINE LOCATE.SECTOR	1
184	ROUTINE MAIN2	1
185	ROUTINE MIN.MOVE	1
186	ROUTINE MPDB.INPUT	1
187	ROUTINE MRT.TO.FREQ	1
188	ROUTINE NORMAL.F	1
189	ROUTINE ORD.MOVCOR	1
190	ROUTINE OUTPUT.EXPENDITURES	1

Figure 3-10  
Modules Ranked by Maximum IF Depth





can suspend, that is let simulated time elapse, and then restart. There is no guarantee that the conditions that were true prior to a suspension are still true afterward.

### 3.5.2 Operand Complexity Metric

In order to determine the operand complexity (Halstead Length Metric) of the COSAGE source code, each module was examined for the number of operands and operators. The number of operands was gathered in a semi-automated fashion by scanning the code and marking the occurrences of each operator mentioned in Section 2.3; namely, +, -, \*, /, >, <, =, ≠, \*\*, ADD, and SUBTRACT. Additionally, phrases such as UNLESS...IS EMPTY, FOR EVERY, NONE imply a relational operator and were included in the count. In the example given in Figure 3.11, there are 24 operators. (Note: The "ADD" on line 3814 was not included in the count since this line of code was added by SAI as part of the invocation study.)

The number of operands per routine was gathered by manually counting the number of occurrences of line number references that is produced by SIMSCRIPT upon compilation of the COSAGE source code. All references were counted and included the following:

- Labels
- Global variables
- Recursive variables
- Define to means
- Routines
- Arguments
- Sets
- Temporary attributes
- Permanent attributes
- Implied subscripts
- Permanent entities
- Process notices
- Function attributes



Options = 10-SUSCHK,TRACE2,OPTIMIZE,TERMINAL,NOLIST,NAME31 5-JAN-1984 12:01 Page ( 1 ) 34

```

ROUTINE UNIT ASSIGNMENT
  GIVEN
    SIDE,
    UNITS,
    TYPE,BATTLE.FIELD,
    THETA,
    X,
    Y
  ADD 1 TO ANAL.CTR(12,1) --- ADVN_ANAL

  NORMALLY MODE IS INTEGER
  DEFINE ANGLE, THETA, THETA.ONE AS REAL VARIABLES
  DEFINE UNITS AS AN INTEGER, 1-DIMENSIONAL ARRAY

  LET NO.UNITS @ DIM.F(UNITS(0))
  LOOP FOR I = 1 TO NO.UNITS
    (UNLESS TEAM.TYPES(TTYPE,BATTLE.FIELD) IS EMPTY)
    DO THE FOLLOWING
      (FOR EVERY TT OF TEAM.TYPES(TTYPE,BATTLE.FIELD)
        WITH TT.IN.USE(TT) @ NO AND
        TT.SIZE(TT) @ SIDE AND
        TT.TYPE(TT) @ UN-UNIT(UNITS(I))
        FIND THE FIRST CASE
        (IF NOW)
          LIST ATTRIBUTES OF TYPE-TEAM CALLED TT
          LIST UN-UNIT(UNITS(I)), SIDE, I, UNITS(I)
          CALL F2002.STOP
          ALWAYS

  LET TT.IN.USE(TT) @ YES
  LOOP (FOR EVERY PP OF PATH.SLIST)
    (UNLESS PATH.SPT(TT) IS EMPTY)
    DO THE FOLLOWING
      CREATE A POINT
      CALL ANGLE.COMPUTE
      GIVEN
        PP.X.POINT(PP),
        PP.Y.POINT(PP)
      YIELDING
        THETA.ONE
      LET ANGLE @ THETA @ THETA.ONE
      LET A @ (PP.Y.POINT(PP))^2 @ PP.X.POINT(PP)^2 @ 2 @ PP.X.POINT(PP) @ PP.Y.POINT(PP) @ 2 @ 0.5 --- OPTIMIZE
      LET P.X(POINT) @ A @ P @ COS.F(ANGLE)
      LET P.Y(POINT) @ Y @ P @ SIN.F(ANGLE)
      FILE THE POINT IN THE UN-UNIT(UNITS(I))
    ENDOLOOP
  ENDOLOOP

  EXITROUTINE
  LERROUTINE

```

Figure 3.11  
Counting Operators for the Halstead Length Metric



In the example shown in Figure 3.12, the total number of operands is 93.

The Halstead complexity is the sum of the number of operators (24) and the number of operands (93). For the examples shown in Figures 3.11 and 3.12, the Halstead length metric is 117.

As with the control complexity metric, the number of operators and operands were tallied using an SAI-developed configuration control form. An example form is shown in Figure 3.4. A post-processor was written to list the COSAGE source modules ranked by Halstead Length. The results of this post-processor are shown in Figure 3.13.

As mentioned previously, if the Halstead length metric of a code module is 270 or greater, it is likely that the module was not properly designed with respect to module/submodule allocation. It is also likely that the module will be difficult to debug and might be of poor programming quality.

Referring to Figure 3.13, 57 COSAGE modules have a Halstead length of 270 or more. This represents approximately 22% of the COSAGE modules. Of these 57, 33 are routines (representing approximately 17% of all COSAGE routines), 16 are processes (representing approximately 84% of all COSAGE processes), 7 are events (representing approximately 2% of all COSAGE events), and 1 is a function (representing approximately 9% of all COSAGE functions).

Further investigation of the processes in the COSAGE model is obviously necessary since 84% of the COSAGE processes have a Halstead length of 270 or above. Recommended changes to

Counting UNIT-ASSIGNMENT  
Options = IO, SUBCMR, TRACF2, OPTIMIZE, TERMINAL, NOLIST, NAME31 5-JAN-1984 12:01 Page 35 ( 1 )

C O U N T I N G O P E R A N D S

NAME	TYPE	MODE	LINE NUMBERS OF REFERENCES
ANAL-CTR	GLOBAL VARIABLE	ASG 379 (2-D)	3614*
ANGLE	RECURSIVE VARIABLE	WORD 1	3617 3646 3648 3649
ANGLE-COMPUTE	ROUTINE	INTEGER	3640
COS-F	ROUTINE	DOUBLE	3648
DIM-F	ROUTINE	INTEGER	3620
ENDLOOP	DEFINE TO MEAN	3651	3652
ENDROUTINE	DEFINE TO MEAN	3655	
ERROR-STOP	ROUTINE	INTEGER	3652
EXITROUTINE	DEFINE TO MEAN	3654	
F-PATH-SET	TEMPORARY ATTRIBUTE	3630	
I	RECURSIVE VARIABLE	WORD 6	3621*
L-PATH-SET	TEMPORARY ATTRIBUTE	3630	3625 3631* 3650
LOOP	DEFINE TO MEAN	3621	3636
M-TEAM-TYPES	TEMPORARY ATTRIBUTE	3630	
N-PATH-SET	TEMPORARY ATTRIBUTE	3630	
NO	DEFINE TO MEAN	3625	
NO-UNITS	RECURSIVE VARIABLE	WORD 5	3621
P-TEAM-TYPES	TEMPORARY ATTRIBUTE	3630	
P-K	TEMPORARY ATTRIBUTE	3648	
P-Y	TEMPORARY ATTRIBUTE	3649	
PATH-SET	SET	3636	3637
POINT	TEMPORARY ENTITY	3639	
PP	GLOBAL VARIABLE	ASG 555	
PP-X-POINT	RECURSIVE VARIABLE	WORD 20	3648 3649 3650
PP-Y-POINT	TEMPORARY ATTRIBUTE	AUTO 1	3642 3643 3647*
S	TEMPORARY ATTRIBUTE	AUTO 2	3647
S-TEAM-TYPES	RECURSIVE VARIABLE	WORD 22	3647 3648 3649
SIDE	TEMPORARY ATTRIBUTE	AUTO 9	3648
SIN-F	TEMPORARY ATTRIBUTE	WORD 1	3625 3631
TEAM-TYPES	ROUTINE	3622	
THETA	SET	(1-0)	3624
THETA-ONE	ARGUMENT	WORD 4	3617 3646
TI	RECURSIVE VARIABLE	WORD 7	3617 3646
TI-IN-USE	TEMPORARY ATTRIBUTE	AUTO 2	3625* 3630 3635
TI-SIDE	TEMPORARY ATTRIBUTE	AUTO 3	3625 3630
TI-TYPE	TEMPORARY ATTRIBUTE	AUTO 4	3625 3630
TI-UNIT	TEMPORARY ATTRIBUTE	AUTO 1	3625 3630
TYPE-BATTLE-FIELD	ARGUMENT	WORD 3	3622 3624
TYPE-TEAM	GLOBAL VARIABLE	ASG 790	
UN-PATH	SET	(1-0)	3630*
UN-TYPE-UNIT	PERMANENT ATTRIBUTE	ASG 795 (1-0)	3625 3631
UNIT-ASSIGNMENT	ROUTINE	3606	
UNITS	ARGUMENT	WORD 2 (1-0)	3618 3620 3625 3631* 3650
X	ARGUMENT	WORD 5	3612 3648
Y	ARGUMENT	WORD 5	3613 3649
YES	DEFINE TO MEAN		3635

Counting Operands for the Halstead Length Metric

Figure 3.12

# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
1	ROUTINE BTRY.EFFECTS	1680
2	PROCESS AC.ATK.TGT	1442
3	PROCESS SHOOT.OUT	1415
4	PROCESS AIR.OBSERVER	1034
5	PROCESS CAS.MISSION	913
6	PROCESS FIRE.MISSION	821
7	PROCESS HELICOPTER.FIRE	877
8	ROUTINE FINAL.COVERAGE	852
9	EVENT AC.ENGAGEMENT	839
10	ROUTINE FLIGHT.PATH	799
11	PROCESS HEL.TARGET.ACQUISITION	791
12	PROCESS FORWARD.OBSERVER	768
13	PROCESS TARGET.REPORT	636
14	ROUTINE SNARE	677
15	ROUTINE AC.DETECTION	674
16	PROCESS ASSESSMENT	647
17	PROCESS HC.ARRIVE.BATTLE	647
18	EVENT START.BATTLE	641
19	ROUTINE EST.COVERAGE	604
20	ROUTINE AC.BOMB.EFFECTS	541
21	ROUTINE AC.DETECTION	515
22	ROUTINE FA.BN.MOVEMENT	482
23	ROUTINE ORIENTATION	465
24	ROUTINE TACAIP.INPUT	451
25	ROUTINE PGM.MSN.ASGN	431
26	ROUTINE EMPLOY.HELICOPTERS	425
27	PROCESS REMOTE.PILOT.VEHICLE	412
28	ROUTINE FA.BN.ASGN	407
29	PROCESS HC.RETURN.FARRP	392
30	ROUTINE AC.OF.EFFECTS	379
31	PROCESS PHOTO.IF.FLIGHT	379
32	ROUTINE ATTRIT.SENSOR	363
33	ROUTINE AC.SHOT	367
34	ROUTINE SENSOR.INPUT	359
35	ROUTINE GENERAL.BATTLE	357
36	EVENT HELO.ENGAGEMENT	351
37	ROUTINE REQUEST.SMOKE	349
38	ROUTINE NEW.SEGMENT	339

Figure 3-13  
Modules Ranked by Halstead Length



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
39	ROUTINE MINE.EFFECTS	338
40	ROUTINE REQUEST.ILLUM	334
41	EVENT OFF.LINE.ATTRITION	333
42	ROUTINE CHECK.CAS.CONSTRAINTS	325
43	PROCESS ARTY.ASSESS	323
44	ROUTINE DEAD.UNIT	322
45	ROUTINE ADJUST	310
46	ROUTINE ANALYSIS.OUTPUT	305
47	PROCESS AIRBORNE.RADAR	296
48	EVENT START.MOVE	295
49	ROUTINE CHECK.FOR.MINES	292
50	ROUTINE UNIT.ENVIR	292
51	FUNCTION HE.WLA	286
52	EVENT UPDATE.LOC	282
53	ROUTINE HC.EMPTY	281
54	EVENT FEGA.SORTIE	276
55	ROUTINE CHANGE.LOC	274
56	ROUTINE ENG.CAS.MISSION	273
57	ROUTINE UNIT.INPUT	272
58	EVENT CBR.OPERATOR	268
59	ROUTINE SNAP.P	267
60	ROUTINE INTER.BATTLE	262
61	PROCESS MINE.ASSESS	262
62	ROUTINE FIND.START.TIME	261
63	ROUTINE PK.COMPUTE	259
64	ROUTINE LINE.OP.SIGHT	257
65	ROUTINE RPV.DETECTION	256
66	ROUTINE CHECK.PROX	252
67	ROUTINE SMOKE.EFFECTS	243
68	ROUTINE FILE.FO.SCHD	245
69	ROUTINE BLOCK.LOS	243
70	ROUTINE HE.OR.ICM.COMPUTATION	234
71	ROUTINE CAS.EVAL	228
72	ROUTINE CBR.DETECTION	222
73	ROUTINE EMPTY	214
74	ROUTINE MARGINAL.EFFECTS.AUJ	214
75	PROCESS WITH.DRAW	213
76	ROUTINE DUST.EFFECTS	212

Figure 3-13  
Modules Ranked by Halstead Length  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
77	ROUTINE HC.DISENGAGE	209
78	ROUTINE MINE.DELAY	208
79	EVENT PDB.OPERATOR	207
80	ROUTINE OUTPUT.EXPENDITURES	206
81	ROUTINE SEARCH.COVERAGE	205
82	FUNCTION ICM.WLA	203
83	ROUTINE MINE.INPUT	201
84	ROUTINE TARGET.ANALYSIS	188
85	ROUTINE INTER.HELO	185
86	ROUTINE PIR.DETECTION	184
87	ROUTINE WHAT.NEXT	184
88	ROUTINE LOCATE.SEARCH.AREA	182
89	ROUTINE MUNS.INPUT	178
90	ROUTINE BTRY.INPUT	177
91	ROUTINE ILLUM.EFFECTS	176
92	ROUTINE BETWEEN.ROUTINE	173
93	ROUTINE KV.PRINT	171
94	EVENT GET.NX.ORD	169
95	ROUTINE PDB.DETECTION	168
96	ROUTINE PREPARE.LIST	162
97	ROUTINE REQUEST.FASCAM	162
98	ROUTINE REQUEST.WD.FASCAM	158
99	EVENT STL.ENDED	155
100	ROUTINE OUTPUT.ATTRITION	155
101	EVENT ACT.ATA	154
102	ROUTINE FEBA.INITIAL	153
103	ROUTINE AMMO.RPT	148
104	ROUTINE WEIGHTED.VOLLEYS	147
105	EVENT START.ARTY.MOVEMENT	147
106	EVENT CFR.ON	145
107	ROUTINE AR.DETECTION	143
108	EVENT ENGAGEMENT	140
109	ROUTINE READ.ORDERS	139
110	ROUTINE FARRP.INPUT	137
111	ROUTINE CHECK.DEAD	136
112	ROUTINE CREATE.FORCE	136
113	ROUTINE TACAIR.DATA.REPORT	136
114	ROUTINE HC.COMPUTE.TIMES	135

Figure 3-13  
Modules Ranked by Halstead Length  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
115	ROUTINE HE.LA.INPUT	130
116	ROUTINE T3F.INPUT	126
117	ROUTINE SIZE.ESTIMATE	126
118	FUNCTION FE3A.BAND	123
119	ROUTINE TIME.TO.DETECT	121
120	ROUTINE SWITCH.F0	118
121	ROUTINE UNIT.ASSIGNMENT	117
122	ROUTINE FILE.KAD.SENSOR	116
123	ROUTINE KV.SCOREBOARD	116
124	ROUTINE LOS.CHECK	116
125	ROUTINE STL.CHECK	110
126	ROUTINE SMOKE.COMPUTATION	108
127	ROUTINE AC.MONS.INPUT	107
128	ROUTINE ENG.MOVE	107
129	ROUTINE CHECK.FORCE	105
130	ROUTINE VOLLEY	105
131	ROUTINE CAT.TU.INPUT	102
132	EVENT ACT.REINF	101
133	ROUTINE FIN.BATTLE	99
134	ROUTINE ILLUM.COMPUTATION	99
135	ROUTINE LINE.CIRCLE	99
136	ROUTINE T3.INPUT	99
137	ROUTINE HEL.RANGE.COMPUTE	97
138	ROUTINE ENG.FE3A.SET	96
139	ROUTINE CFR.DEGRADE	93
140	ROUTINE EQ.TE.INPUT	92
141	ROUTINE NOISE.DEGRADE	92
142	ROUTINE MFO.INPUT	91
143	ROUTINE UNIT.PRIORITY	91
144	ROUTINE COMBINE.TPS	89
145	PROCESS HOW.REPAIR	86
146	ROUTINE PK.INPUT	87
147	ROUTINE FASCAM.COMPUTATION	85
148	ROUTINE PRED.POS	85
149	ROUTINE PROX.CHECK	85
150	EVENT CFR.ACTIVATION	84
151	ROUTINE SYS.INPUT	83
152	EVENT PDB.ACTIVATION	81

Figure 3-13  
Modules Ranked by Halstead Length  
Continued





# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
153	ROUTINE FARRP.CHECK	30
154	ROUTINE PLAT.COUNT	20
155	ROUTINE COPY	79
156	ROUTINE FBN.FD.INPUT	74
157	ROUTINE MAO.INPUT	74
158	ROUTINE REQUEST.DEP.FASCAM	73
159	ROUTINE TT.FACTORS.INPUT	73
160	ROUTINE CHECK.LIST	71
161	FUNCTION BTRY.AVAILABLE	70
162	ROUTINE TYPE.WEAPON.INPUT	70
163	ROUTINE FORM.TF.LIST	69
164	EVENT INIT.PREPLAN.CAS	69
165	ROUTINE COMPUTE.WD	67
166	ROUTINE INIT.REINF	66
167	ROUTINE TEMPERATURE.ATTENUATION	66
168	ROUTINE RESET.FB3A.SECTOR	64
169	EVENT ACT.MOVDIS	63
170	ROUTINE DECIDE	62
171	ROUTINE GAMMA.F	62
172	ROUTINE NORMAL.F	62
173	ROUTINE REM.EFFECTS.COMPUTATION	62
174	ROUTINE CONTRAST.TO.FREQ	61
175	FUNCTION AR.PROB.DETECT	60
176	ROUTINE DESTROY.CRD	60
177	ROUTINE REIN.ARRIVE	60
178	FUNCTION COMBINATIONS	59
179	ROUTINE MAIN2	59
180	ROUTINE PROB.INF	59
181	FUNCTION COLLISION	59
182	EVENT HC.DEPART.BATTLE	58
183	ROUTINE XV.INPUT	58
184	ROUTINE PGM.INPUT	58
185	ROUTINE FRAC.COMPUTE	57
186	ROUTINE REPLACE.HC	57
187	ROUTINE PROB.TIME	56
188	ROUTINE MCFR.INPUT	55
189	ROUTINE COMPARE.TRS	53
190	ROUTINE RUL.EN.INPUT	53

Figure 3-13  
Modules Ranked by Halstead Length  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
191	ROUTINE MPOB.INPUT	52
192	ROUTINE PREP.WITHDRAW	52
193	EVENT STOP.ARTY.MOVEMENT	52
194	EVENT ACT.MOVCOR	51
195	ROUTINE POSITION	51
196	ROUTINE SEARCH	51
197	ROUTINE DEQ.FEBA.SET	50
198	ROUTINE LOCATE.SECTOR	50
199	ROUTINE P.E.M.INPUT	50
200	ROUTINE MIN.MOVE	49
201	ROUTINE SUBM.INPUT	49
202	ROUTINE SMOKE.INPUT	48
203	EVENT CFR.OFF	47
204	ROUTINE TR.INPUT	47
205	ROUTINE COMPUTE.C	44
206	ROUTINE MAOS.INPUT	44
207	ROUTINE ANGLE.COMPUTE	43
208	ROUTINE ORD.MOVCOR	43
209	EVENT ACT.DEF	40
210	ROUTINE CHECK.STREN	39
211	ROUTINE CHK.FD.TR	39
212	ROUTINE CREATE.TEAMS	39
213	ROUTINE ILLUM.INPUT	39
214	ROUTINE PROX.POS	39
215	ROUTINE TERM.CHECK	38
216	ROUTINE FINISH.COMPUTATION	37
217	ROUTINE EST.MIL.WORTH	36
218	FUNCTION EST.RANGE	36
219	ROUTINE MAIN1	36
220	ROUTINE CHK.COMP.TR	35
221	EVENT ARTY.OCCUPATION	34
222	EVENT SET.JEBUG	34
223	EVENT CHANGE.LITE	33
224	FUNCTION EST.TR.RANGE	33
225	ROUTINE POSITION.OUT	33
226	ROUTINE BTRY.FM.DEQ	31
227	ROUTINE FD.EFFECTS.REQ	30
228	EVENT SEND.TEAM	30

Figure 3-13  
Modules Ranked by Halstead Length  
Continued



# SCIENCE APPLICATIONS, INC.

RANK	MODULE NAME	HALSTEAD LENGTH
229	ROUTINE GET.TERRAIN	29
230	ROUTINE RANGE.COMPUTE	29
231	FUNCTION ACT.RANGE	27
232	ROUTINE BTRY.FM.ENQ	27
233	ROUTINE FDC.TR.DEN	27
234	ROUTINE VIS.INPUT	27
235	FUNCTION STAY.TIME	25
236	ROUTINE CHECK.ENGAGEMENT	24
237	EVENT DQ.OLD.SORTIE.QUEUE	24
238	ROUTINE EXPONENTIAL.F	24
239	ROUTINE FDC.TR.ENQ	23
240	ROUTINE INITIAL.MOVE	23
241	ROUTINE MAIN3	23
242	ROUTINE SEGMENT.ADJUST	22
243	ROUTINE DECISION.INPUT	21
244	ROUTINE INITIAL.DETECT	21
245	ROUTINE ST.INPUT	21
246	ROUTINE MRT.TO.FREQ	19
247	ROUTINE WEIBULL.F	19
248	EVENT CHANGE.WEATHER	17
249	EVENT SCHEDULE.ARTY.MOVEMENT	17
250	ROUTINE ORD.UEF	14
251	ROUTINE ORD.MOVDIS	14
252	ROUTINE DQ.CMSN.QUEUE	13
253	ROUTINE ORD.ATK	12
254	ROUTINE ORD.REINF	12
255	EVENT POSITION.REPORT	12
256	EVENT MOVE	11
257	PROGRAM MAIN	10
258	ROUTINE TIME.REQ	10
259	ROUTINE PROXIMITY.REQ	9
260	EVENT END.SIMULATION	8
261	ROUTINE JOHNSON.CRITERIA	7
262	ROUTINE ERROR.STOP	5
263	ROUTINE HEADING	4
264	ROUTINE OPEN.INPUT.OUTPUT.FILES	1
265	PROGRAM OLDER.VERSION.PREAMBLE	0
266	PROGRAM PREAMBLE	0

Figure 3-13  
Modules Ranked by Halstead Length  
Continued



reduce the number of operators and operands in the processes are discussed in Section 4.8 of this report.



#### 4.0 RECOMMENDED CHANGES

This section presents the changes recommended for the COSAGE model. These recommendations were compiled based on SAI's efforts in the static and dynamic analyses.

##### 4.1 Exponentiation

A review of the COSAGE source code shows that there has been widespread use of exponentiation for squaring and cubing mathematical expressions. Figure 4.5 shows an example of this type of calculation. The SIMSCRIPT implementation of exponentiation is not a very efficient means of accomplishing squares or cubes. Figure 4.6 illustrates an enhanced method, which multiplies the local variables (DELTA.X and DELTA.Y) times themselves. This method, when benchmarked on both the VAX and SPERRY computers, averaged an execution speed improvement of ~~approximately 60 percent~~.

When only two of ~~the most frequently~~ invoked COSAGE modules were revised in the manner discussed above and benchmarked on the VAX version of COSAGE, system execution time decreased by ~~approximately 10 percent~~ improvement. (Details of this benchmark are discussed more fully in Section 4.7 of this report.) Based on this test case, SAI analysts concluded that the exponentiation optimization holds a very high potential for reducing COSAGE execution time. Although it is difficult to estimate the precise

The logo for Science Applications, Inc. (SAI) is located in the bottom right corner of the page. It consists of the letters 'SAI' in a stylized, bold, italicized font.

savings which can be recognized by this revision, there are more than one hundred calculations in COSAGE which use exponentiation. Therefore, it is realistic to expect an overall execution time savings in the range of 5-10% when all COSAGE calculations using exponentiation have been upgraded to use the demonstrated technique.

#### 4.2 Inefficient Mathematical Expressions

Writing source code is frequently a tradeoff between clarity and efficiency. Because of the unusual units of measure (e.g., hexadecameters), conversion of expressions are often performed with factors such as (16.0/10.0) or (10.0/16.0). These factors make the code clearer, but are extremely inefficient since they must be re-evaluated every time they are executed. This type of factor is found at several hundred locations in the COSAGE source code.

SAI analysts recommend replacing the inefficient expressions with pre-calculated global variables using meaningful names. For example, the SIMSCRIPT statement:

```
LET TEN.16THS = 10.0/16.0
```

would allow all the expressions (10.0/16.0) to be replaced with a meaningful variable (TEN.16THS) containing the same value. This optimization would reduce both the execution time and the memory requirements.



#### 4.3 Unnecessary SQRT.F Usage

At many places in the COSAGE source code, the distance between two points is calculated using the square root of the sum of the delta X squared and delta Y squared.

When the actual distance between the points is required, this type of algorithm is relatively efficient; but often, the actual distance is not required. The objective may be to select the closest alternative to a particular location, in which case the square root is not required.

When distances are being compared to a threshold or range, it is much more efficient to square the threshold or range once and compare the sum of squares to this value, than to take the square root many times to compare the actual distances. Benchmarks performed by SAI analysts indicate identical results can be obtained with 30% to 65% less execution time required.

#### 4.4 Schedules/Reschedules

COSAGE contains many events which are scheduled to occur at various points in simulated time. Some events schedule re-occurrences of the same event at a later instance in simulated time. This type of event is best illustrated by the periodic update of location that can occur at regular intervals for moving units.

The SIMSCRIPT compiler by default automatically deallocates the memory used by the event notice just before the event is executed. For these types of periodic events, the optional phrase "SAVING THE EVENT NOTICE" should be appended to



the EVENT statement. Then SIMSCRIPT will allow the event notice to be re-used.

The re-use is accomplished by replacing the repeated "SCHEDULE A" statement with a "RESCHEDULE THIS" statement. The overhead savings for frequently used events can be substantial. SAI analysts wrote two programs to test the efficiencies of replacing repeated schedule statements with the reschedule option.

The program which appears in Figure 4.1 schedules an event which in turn schedules itself again at 1 hour intervals over a period of 1000 hours. The program which appears in Figure 4.2 is identical in every way to the first program except that the event notice is saved and the event reschedules itself. The elapsed CPU time savings are summarized below.

24.08	seconds	(SCHEDULE)
<u>10.16</u>	seconds	(RESCHEDULE)
12.90	seconds	(54% savings)

There are several places which have been identified in the COSAGE source code where SCHEDULE statements should be replaced with RESCHEDULE ones, and the event notices should be saved. The events identified include:

CPR.OPERATOR  
CHANGE.LITE  
FEBA.SORTIE  
PDB.OPERATOR  
POSITION.REPORT  
SCHEDULE.ARTY.MOVEMENT  
UPDATE.LOC

In addition to saving execution time, this recommended change also has the advantage of saving memory since the previously allocated space is reused, and no new space is required.

#### 4.5 Removal/Replacement of Identified Modules





```

PROGRAM PREAMBLE
  EVENT NOTICES INCLUDE SCHEDULE_E
END

PROGRAM MAIN
  SCHEDULE AN SCHEDULE_E IN 1 HOUR
  START SIMULATION
END

EVENT TO SCHEDULE_E
  SCHEDULE AN SCHEDULE_E IN 1 HOUR
  IF TIME.V > 1000
    STOP
  OTHERWISE
    RETURN
END

```

Figure 4.1  
Schedule Testcase



```

**PROGRAM** PREAMBLE
  EVENT NOTICES INCLUDE SCHEDULE_E
END

**PROGRAM** MAIN
  SCHEDULE AN SCHEDULE_E IN 1 HOUR
  START SIMULATION
END

EVENT TO SCHEDULE_E SAVING THE EVENT NOTICE
  RESCHEDULE A SCHEDULE_E IN 1 HOUR
  IF TIME.V > 1000
    STOP
  OTHERWISE
    RETURN
END
```

Figure 4.2  
Reschedule Testcase



There are eleven modules identified in the accompanying SAI-SDDL processed COSAGE source code (see Volume II) which have been categorized as un-used and/or deletion candidates. Some of the modules simply return a constant value; they should be replaced by a global variable. Some modules are not currently implemented in the program. Each should be evaluated to determine where removal/replacement should be performed. A recommended action is listed for each in Table 4.1.

One such module (JOHNSON.CRITERIA) was invoked 344,157 times in COSAGE. This routine simply returns a value of 1.0. A testcase was written which replicates 344,157 calls (see Figure 4.3). Another testcase was rewritten which simply assigns a variable the value of 1.0 (see Figure 4.4). The results are summarized below:

13.59	CPU seconds for Call Statements
<u>3.51</u>	CPU seconds for Assignment Statements
10.08	Savings (74%)

#### 4.6 Utilize SIMSCRIPT Text Feature

Utilization of the recently-implemented SIMSCRIPT text feature is recommended. The replacement of alpha variable types with text variable types will serve two purposes: efficiency of memory usage and transportability of the source code.

Alpha variables or constants are left justified when stored in a computer word. It depends on the implementation for the particular machine whether a single character or more is stored per word. Regardless of the implementation, however, if fewer characters are stored per word than the number of bytes in the computer word, storage is wasted.

Text variables, on the other hand, regardless of implementation, are represented by a pointer giving the address of a memory location where one or more words contain the represented characters, one character per byte.

<u>Module Name</u>	<u>Comments</u>
GAMMA.F	Not used - Delete
AIRBORNE.RADAR	Not used - Delete
AR.DETECTION	Called by AIRBORNE.RADAR (not used) - Delete
AR.PROB.DETECTION	Called by AR.DETECTION (not used) - Delete
PHOTO.IR.FLIGHT	Not used - Delete
STAY.TIME	Called by PHOTO.IR.FLIGHT (not used) - Delete
JOHNSON.CRITERIA	Returns a Constant Value (1.0) Called 344,157 times (5% of all invocations) 33rd most frequently sampled Should be replaced by a Global Variable
PROXIMITY.REQ	Returns a constant value (5) Should be replaced by a Global Variable
TIME.REQ	Returns a constant value (0.1) Should be replaced by a Global Variable
OLDER VERSION PREAMBLE	Should be deleted from file to avoid confusion and errors
PLAI.COUNT	Only calls are from event START.BATTLE Calls are commented out Remove comments and delete routine

TABLE 4.1

Modules To Be Deleted/Replaced



```
..

"PROGRAM" MAIN

  DEFINE I AS AN INTEGER VARIABLE
  FOR I = 1 TO 344157
  DO
    CALL JOHNSON.CRITERIA YIELDING .NO.BARS
  LOOP
END

ROUTINE JOHNSON.CRITERIA YIELDING .NO.BARS
  LET .NO.BARS = 1.
  RETURN
END
```

Figure 4.3  
344,157 Call Statements Testcase



```
PROGRAM MAIN  
  DEFINE I AS AN INTEGER VARIABLE  
  LET .NO.BARS = 1.  
  FOR I = 1 TO 344157  
  DO  
    LET M = .NO.BARS  
  LOOP  
END
```

Figure 4.4  
344,157 Assignment Statements Testcase



Further, when code is transported from one machine to another, it must first be determined if the implementations of alpha modes is compatible before alpha variables may be used with confidence. The convention of using text modes instead of alpha mode would be both more efficient and would increase transportability of code.

#### 4.7 Perform a Thorough Analysis on the 26 Most Frequently Invoked Modules

SAI dynamic execution analysis has shown the following group of 26 modules (10%) of the COSAGE program to account for over 93% of all invocations (see Figure 3.1). It is recommended that SAI analyze each of these 26 modules in detail and in close co-operation with the COR. Small individual changes in efficiency can result in large overall savings since these modules are invoked frequently.

For example, the two program modules most frequently invoked were FUNCTION ACT.RANGE (1,189,098 invocations) and ROUTINE RANGE.COMPUTE (792,643). Together, these two modules account for over 29% of all invocations in the baseline 24 hour COSAGE simulation. Both modules had been highlighted by SAI analysts during our static analysis with the \OPTIMIZE cross-reference identifier.

The first module, ACT.RANGE, Figure 4.5, computes the intermediate values DELTA.X and DELTA.Y; the values are squared, summed, and used as an argument to the SIMSCRIPT SQRT.F function. This method of squaring the values is exponentiation; during static analysis benchmarks, analysts found this method of squaring to require up to twice as much execution time with SIMSCRIPT.

```
FUNCTION ACT.RANGE                                *** F001
  GIVEN
    UNIT1,
    UNIT2

  ADD 1 TO ANAL.CTR(239,1) *** \DYN_ANAL
  'THIS FUNCTION COMPUTES THE ACTUAL RANGE BETWEEN TWO UNITS

  NORMALLY MODE IS INTEGER
  DEFINE RANGE AS A REAL VARIABLE

  LET DELTA.X = UN.X.COORD( UNIT1 ) - UN.X.COORD( UNIT2 )
  LET DELTA.Y = UN.Y.COORD( UNIT1 ) - UN.Y.COORD( UNIT2 )
  LET RANGE = SQRT.F( DELTA.X ** 2 + DELTA.Y ** 2 ) *** \OPTIMIZE

  RETURN WITH RANGE
ENDFUNCTION
```

Figure 4.5  
Current Function ACT.RANGE



Therefore, we recommend replacing this function with the code indicated in Figure 4.8.

The second module, RANGE.COMPUTE, Figure 4.7, performs the same function with three basic differences: 1) the module is a routine, not a function (this only affects how it is invoked and used), 2) it uses real intermediate variables and returns an integer answer, and 3) the square-root is implemented via exponentiation. Benchmark runs indicate this method of finding the square root requires approximately 70% more execution time. Augmented with the same method of squaring proposed for ACT.RANGE, the recommended revision is shown in Figure 4.8.

Benchmarks when these two revised routines were implemented in SAI's VAX Virtual Test Suite for COSAGE show a decrease of more than 2.7% in execution time. A similar savings would be reasonable to expect with the SPERRY SIMSCRIPT version of COSAGE. Further, examination of the calling locations shows that two modules which perform the same purpose (one yielding an integer result, the other a real result) are not necessary. Any required mode conversions can be performed after the call. This integration would decrease the memory requirement as well and provide a uniform, efficient approach to fulfilling a single function. Comparisons made using these inconsistent methods may result in unexpected program behavior.

#### 4.8 Modularize Candidate Processes

The following COSAGE processes were identified because they have source code line counts in excess of 120 (approximately 2 pages):

AC.ATK.TGT  
AIR.OBSERVER

```

FUNCTION ACT.RANGE
  GIVEN UNIT1 AND UNIT2                                ''~ F001

  ADD 1 TO ANAL.CTR(239,1) ''~ \DYN-ANAL
  ''THIS FUNCTION COMPUTES THE ACTUAL RANGE BETWEEN TWO UNITS

  DEFINE UNIT1, UNIT2, DELTA.X, AND DELTA.Y AS INTEGER VARIABLES
  DEFINE RANGE AS A REAL VARIABLE

  LET DELTA.X = UN.X.COORD( UNIT1 ) - UN.X.COORD( UNIT2 )
  LET DELTA.Y = UN.Y.COORD( UNIT1 ) - UN.Y.COORD( UNIT2 )
  LET RANGE = SQRT.F(REAL.F(DELTA.X*DELTA.X+DELTA.Y*DELTA.Y)) ''~ \OPTIMIZE

  RETURN WITH RANGE

END

```

Figure 4.6

Proposed Function ACT.RANGE

ROUTINE RANGE.COMPUTE

\*\*\* C018

GIVEN

UNIT.A,

UNIT.B

YIELDING

RANGE.

ADD 1 TO ANAL.CTR(129,1) \*\*\* \DYN-ANAL

NORMALLY MODE IS INTEGER

DEFINE D.X., D.Y. AS REAL VARIABLES

LET D.X. = UN.X.COORD(UNIT.A)-UN.X.COORD(UNIT.B)

LET D.Y. = UN.Y.COORD(UNIT.A)-UN.Y.COORD(UNIT.B)

LET RANGE = (D.X\*\*2+D.Y\*\*2)\*\*.5 \*\*\* \OPTIMIZE

EXITROUTINE

ENDROUTINE

Figure 4.7  
Current Routine RANGE.COMPUTE



```

ROUTINE RANGE.COMPUTE                                *** C018
  GIVEN UNIT.A AND UNIT.B
  YIELDING RANGE

  ADD 1 TO ANAL.CTR(129,1)  *** \DYN_ANAL
  DEFINE D.X., D.Y. AS REAL VARIABLES
  DEFINE UNIT.A, UNIT.B, AND RANGE AS INTEGER VARIABLES

  LET D.X. = UN.X.COORD(UNIT.A)-UN.X.COORD(UNIT.B)
  LET D.Y. = UN.Y.COORD(UNIT.A)-UN.Y.COORD(UNIT.B)
  LET RANGE = SQRT.F(D.X.*D.X. + D.Y.*D.Y.)  *** \OPTIMIZE

  RETURN
END

```

Figure 4.8  
Proposed Routine RANGE.COMPUTE



AIRBORNE.RADAR  
ARTY.ASSESS  
ASSESSMENT  
CAS.MISSION  
FIRE.MISSION  
FORWARD.OBSERVER  
HC.RETURN.FARRP  
HELICOPTER.FIRE  
HEL.TARGET.ACQUISITION  
HC.ARRIVE.BATTLE  
REMOTE.PILOT.VEHICLE  
SHOOT.OUT  
TARGET.REPORT

These 15 processes represent 84% of all COSAGE processes. All of these processes have a Halstead length of 270 or greater. The two remaining processes, PHOTO.IR.FLIGHT and WITH.DRAW have 115 and 112 source lines respectively and Halstead lengths of 379 and 213.

SAI analysts recommend modularizing at least the top 15 processes (ranked by number of source lines) to increase understandability and maintainability. A procedure similar to the following one is recommended.

- First, in close conjunction with the COR, identify a high-level system of comments. An example, using FIRST-NEXT comments is shown in Figure 4.9.
- Next, identify which comment blocks can be modularized and moved out-of-line (into separate routines).

```

23 ROUTINE LOS.CHECK **
24 GIVEN UNIT
25
26 *****
27 * THIS ROUTINE CHECKS TO SEE IF LINE-OF-SIGHT STILL EXISTS AT THIS POINT
28 *****
29 *****
30 FIRST, DEFINE VARIABLES
31 DEFINE UNIT, FORCE, OTHER.UNIT, RANGE, RNG. AS INTEGER VARIABLES
32
33 NEXT, LOCATE THE CLOSEST ENEMY FORCE
34 FOR EVERY FORCE OF HTL.FORCE.SET(UN.BATTLE.INDEX(UNIT))
35 WITH FR.SIDE(FORCE) NE UN.COLOR(UNIT)
36 FIND THE FIRST CASE
37 IF NONE,
38   CALL ERROR.STOP
39 ENUIF
40 LOOP

```

PAGE 1  
LOS\$CHECK  
----- \1>(600)



Figure 4.9  
Example of High-level Comment  
System and Code Standardization

```

41 NEXT, FIND A UNIT IF THIS ONE IS NOT VISIBLE
42 FOR EVERY OTHER UNIT OF PR.UNIT.SET(FORCE)
43 DO THE FOLLOWING,
44 FOR EVERY VISIBLE UNIT OF UN.LOS.LIST(OTHER.UNIT)
45 WITH VU.POINTER(VISIBLE.UNIT) = UNIT
46 FIND THE FIRST CASE
47 IF NONE,
48 IF UN.STATUS(UNIT) = STATIONARY OR UN.STATUS(UNIT) = STA.TO.WITH
49 IF UN.STATUS(OTHER.UNIT) = STATIONARY OR UN.STATUS(OTHER.UNIT) = STA.TO.WITH
50 LET RNG. = TT.STATIONARY.LOS.BREAK(BTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
51 ELSE
52 LET RNG. = TT.M.S.LOS.BREAK(BTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
53 ALWAYS
54 ELSE
55 IF UN.STATUS(OTHER.UNIT) = STATIONARY OR UN.STATUS(OTHER.UNIT) = STA.TO.WITH
56 LET RNG. = TT.M.S.LOS.BREAK(BTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
57 ELSE
58 LET RNG. = TT.MOVING.LOS.BREAK(BTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
59 ALWAYS
60 ALWAYS
61 CALL RANGE.COMPUTE----->>>(337)
62 GIVEN UNIT, OTHER.UNIT
63 YIELDING RANGE
64 IF RANGE LE FND.
65 CREATE A VISIBLE.UNIT
66 LET VU.POINTER(VISIBLE.UNIT) = UNIT
67 LET VU.STATUS(VISIBLE.UNIT) = NO
68 FILE THIS VISIBLE.UNIT IN THE UN.LOS.LIST(OTHER.UNIT)
69 CREATE A VISIBLE.UNIT
70 LET VU.POINTER(VISIBLE.UNIT) = OTHER.UNIT
71 LET VU.STATUS(VISIBLE.UNIT) = NO
72 FILE THIS VISIBLE.UNIT IN THE UN.LOS.LIST(UNIT)
73 ALWAYS
74 ALWAYS
75 CALL CHECK.ENGAGEMENT----->>>(129)
76 GIVEN OTHER.UNIT
77 ENULOGP
78
79 LAST, CHECK TO SEE IF UNIT IS ALREADY ENGAGED
80 CALL CHECK.ENGAGEMENT----->>>(129)
81 GIVEN UNIT
82 -----EXITROUTINE
83
84 ENROUTINE

```

Figure 4.9

Example of High-level  
System and Code Standardization  
(continued)



- Next, perform metric analyses on identified submodules.
- Next, move submodules into separate routines and make required coding changes.
- Last, test modularized processes.

#### 4.9 Standardize the COSAGE Source Code

While performing the various analyses of the COSAGE source code, it became apparent to the SAI analysts that numerous, varying coding conventions and styles had been used in the model development. These inconsistencies made it difficult to read and understand the SIMSCRIPT source code; they also represent inefficiencies in COSAGE. Therefore, it is recommended that a consistent set of coding standards be developed and applied to the COSAGE source code. Other standardization issues should also be addressed; these issues include:

- Examining SIMSCRIPT DEFINE-TO-MEAN statements in COSAGE to determine if they are required or in need of modification.
- Checking for redundant NORMALLY statements.
- Deleting SIMSCRIPT comment statements which are obsolete or unclear.
- Developing a system of high-level comments, with the assistance of CAA personnel knowledgeable of the COSAGE model, which provides insight into the operations/functions being performed by a block of source code.



- Verifying that output units are consistent between routines.
- Developing an @ADD file for the SPERRY which will process the COSAGE source code with the SAI-SDDL processor to automate the production of up-to-date documentation.

Figure 4.10 represents a current COSAGE routine; Figure 4.9 is the same routine which has been updated with the recommended coding standards and processed by SAI-SDDL.

#### 4.10 Develop Graphical Input/Output Capabilities

It is recommended that graphical input/output capabilities be developed for the COSAGE model. The COSAGE model requires voluminous input data. This data requires considerable time as well as in-depth knowledge of the COSAGE program to prepare. The existing EDITS program provides a data checking capability; however, it is recommended that an enhanced data preparation tool be developed. A possible scenario for such an input generation tool would allow a COSAGE user to graphically configure units on a specified terrain, and then have the input processor automatically generate the coordinates, equipment lists, etc. Additionally, this tool could check for typical input data errors (like the ones listed in Section 3.4.1 of this report). Likewise, the development of a graphical output is recommended. Such a tool could display unit movement, attrition, etc.

SAI has developed the Tactics, Operations, and Planning \* Station (TOPS\*). TOPS is a minicomputer-based, color graphics system based on digital map technology. Preliminary investigation indicates that this graphics system could provide input/output

\* A trademark of Science Applications, Inc.



```

1  ROUTINE FOR CHECK
2  GIVE
3  UNIT
4  *LOS CHECK
5  NORMALLY DONE IN INTEGER
6  FOR EVERY FORCE OF HTL.FORCL.SET(UN.BATTLE.INDEX(UNIT))
7  WITH FR.CID(FORCL) = UN.COLOR(UNIT)
8  FIND THE FIRST CASE
9  IF NONE, CALL FRMR.STOP
10  IF CL
11  LOOP
12  FOR EVERY OTHER UNIT OF FR.UNIT.SET(FORCE)
13  DO THE FOLLOWING
14  FOR EVERY VISIBLE UNIT OF UN.LOS.LIST(OTHER.UNIT)
15  WITH VU.POINTER(VISIBLE.UNIT) = UNIT
16  FIND THE FIRST CASE
17  IF NONE
18  IF UN.STATUS(UNIT) = STATIONARY
19  OR UN.STATUS(UNIT) = STA.TO.WITH
20  IF UN.STATUS(OTHER.UNIT) = STATIONARY
21  OR UN.STATUS(OTHER.UNIT) = STA.TO.WITH
22  LET ENG. = IT.STATIONARY.LOS.BREAK(HTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
23  ELSE
24  LET ENG. = IT.M.S.LOS.BREAK(HTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
25  ALWAYS
26  ELSE
27  IF UN.STATUS(OTHER.UNIT) = STATIONARY
28  OR UN.STATUS(OTHER.UNIT) = STA.TO.WITH
29  LET ENG. = IT.M.S.LOS.BREAK(HTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
30  ELSE
31  LET ENG. = IT.MOVING.LOS.BREAK(HTL.TERRAIN.TYPE(UN.BATTLE.INDEX(UNIT)))
32  ALWAYS
33  ALWAYS
34  CALL FRMR.COMPUTE
35  GIVE
36  UNIT,
37  OTHER.UNIT
38  YIELDING
39  RANGE
40  IF RANGE LE ONE
41  CREATE A VISIBLE UNIT
42  LET VU.POINTER(VISIBLE.UNIT) = UNIT
43  LET VU.STATUS(VISIBLE.UNIT) = NO
44  FILE THIS VISIBLE UNIT IN THE UN.LOS.LIST(OTHER.UNIT)
45  CREATE A VISIBLE UNIT
46  LET VU.POINTER(VISIBLE.UNIT) = OTHER.UNIT
47  LET VU.STATUS(VISIBLE.UNIT) = NO
48  FILE THIS VISIBLE UNIT IN THE UN.LOS.LIST(UNIT)
49  ALWAYS
50  ALWAYS
51  CALL CHECK.ENGAGEMENT
52  GIVEN
53  OTHER.UNIT
54  ENDLOOP
55  CALL CHECK.ENGAGEMENT
56  GIVE
57  UNIT
58  EXITROUTINE
59  ENROUTINE

```

FIGURE 4-10 CURRENT COSAGE ROUTINE

processors for simulation models such as COSAGE. Therefore, it is recommended that the linkage between COSAGE and TOPS be studied.



## 5.0 PROPOSED PREAMBLE

Since the PREAMBLE of a SIMSCRIPT program serves as a definition of data structures and of program events and processes, SAI analysts conducted an analysis of it in order to identify areas which could be optimized or updated in order to increase its clarity and maintainability. This section presents the specific observations made and changes recommended.

### 5.1 Existing Structure

Examination of the COSAGE PREAMBLE indicated that the prevailing scheme of organization is alphabetization (see Figure 5.1). However, this scheme does not appear to be rigorously followed. Further, data structures are usually grouped into categories such as permanent entities, temporary entities, processes, events, global variable, set, array, and function definitions, and substitutions.

### 5.2 Proposed Structure

SAI's analysts recommend restructuring the COSAGE PREAMBLE using a hierarchical scheme for organizing the permanent and temporary entities, sets, and attribute definitions. An example of the recommended hierarchical structure is shown in Figure 5.2. Such a scheme should provide more clarity into data structure relationships.



## SCIENCE APPLICATIONS, INC.

EVERY AO.RANGE.BAND HAS  
AN AO.RB.RANGE,  
BELONGS TO  
THE AO.RB.SET

HAS

A P.AO.RB.SET,  
A S.AO.RB.SET,  
A M.AO.RB.SET

EVERY BTRY HAS

A BY.BN,  
A BY.STATUS,  
A BY.TYPE,  
A BY.PGM.FM,  
A BY.CUR.FM,  
A BY.N.ROUNDS,  
A BY.UNIT,  
A BY.BN.RANK,  
A BY.FIRE.RATE,  
A BY.PGM.CAP,  
A BY.STOP.FASCAM.SUPP "TIME.V \* 60

OWNS

A BY.HOW.SET,  
A BY.SCHD.LIST,  
A BY.FM.QUEUE

BELONGS TO

A BN.BTRY.SET

HAS

A F.BY.HOW.SET,  
A L.BY.HOW.SET,  
A F.BY.SCHD.LIST,  
A L.BY.SCHD.LIST,  
A F.BY.FM.QUEUE,  
A L.BY.FM.QUEUE,  
A P.BN.BTRY.SET,  
A S.BN.BTRY.SET,  
A M.BN.BTRY.SET,  
A N.BY.HOW.SET,  
A N.BY.SCHD.LIST,  
A N.BY.FM.QUEUE

Figure 5.1  
Existing PREAMBLE Scheme

EVERY CATEGORY HAS

A CT.NAME,  
A CT.GROUP,  
A CT.MIN.FEBA

OWNS

A CT.TU.SET

BELONGS TO

A GP.CAT.SET

HAS

A F.CT.TU.SET,  
A L.CT.TU.SET,  
A N.CT.TU.SET,  
A P.GP.CAT.SET,  
A S.GP.CAT.SET,  
A M.GP.CAT.SET

EVERY CATEGORY, DIST.FROM.FEBA.BAND, IC.MUNITION HAS  
A CDI.USAGE.INDICATOR

EVERY CATEGORY, DIST.FROM.FEBA.BAND, TYPE.BTRY HAS



# SCIENCE APPLICATIONS, INC.

## PERMANENT ENTITIES

EVERY FA.BN HAS

A FB.MISSION,  
A FA.BN.UNIT

OWNS

A BN.BTRY.SET

HAS

A F.BN.BTRY.SET,  
A L.BN.BTRY.SET,  
A N.BN.BTRY.SET

DEFINE FB.MISSION AS A TEXT VARIABLE

EVERY BTRY HAS

A BY.BN,  
A BY.STATUS,  
A BY.TYPE,  
A BY.PGM.FM,  
A BY.CUR.FM,  
A BY.N.ROUNDS,  
A BY.UNIT,  
A BY.BN.RANK,  
A BY.FIRE.RATE,  
A BY.PGM.CAP,  
A BY.STOP.FASCAM.SUPP \*\*TIME.V \* 60

OWNS

A BY.HOW.SET,  
A BY.SCHD.LIST,  
A BY.FM.QUEUE

BELONGS TO

A BN.BTRY.SET

HAS

A F.BY.HOW.SET,  
A L.BY.HOW.SET,  
A F.BY.SCHD.LIST,  
A L.BY.SCHD.LIST,  
A F.BY.FM.QUEUE,  
A L.BY.FM.QUEUE,  
A P.BN.BTRY.SET,  
A S.BN.BTRY.SET,  
A M.BN.BTRY.SET,  
A N.BY.HOW.SET,  
A N.BY.SCHD.LIST,  
A N.BY.FM.QUEUE

DEFINE BN.BTRY.SET AS A SET RANKED BY LOW BY.BN.RANK

DEFINE BY.BN.RANK, BY.FIRE.RATE AND BY.STOP.FASCAM.SUPP AS SIGNED INTEGERS

## TEMPORARY ENTITIES

EVERY HOW HAS

A HW.BTRY, \*\*OWNING BTRY  
A HW.SFAIL.RNDS, \*\*ROUNDS TILL SHORT TERM FAILURE  
A HW.LFAIL.RNDS, \*\*ROUNDS TILL LONG TERM FAILURE

BELONGS TO

A BY.HOW.SET \*\*WHEN NOT FAILED

HAS

A P.BY.HOW.SET,  
A S.BY.HOW.SET,  
A M.BY.HOW.SET

DEFINE BY.HOW.SET AS A LIFO SET

DEFINE HW.BTRY, HW.SFAIL.RNDS, AND HW.LFAIL.RNDS AS SIGNED INTEGERS

Figure 5.2

Recommended Hierarchical PREAMBLE Scheme



It is also suggested that sections like the events and substitutions be re-alphabetized. This will make it easier to find names which have already been used, since inadvertant reusage could cause errors in COSAGE that would be difficult to trace.

Another recommended PREAMBLE change is to replace inefficient define to means, such as:

```
DEFINE NORTH TO MEAN PI.C/2  
DEFINE SOUTH TO MEAN 3.*PI.C/2
```

with statements like:

```
DEFINE NORTH TO MEAN 1.5707963  
DEFINE SOUTH TO MEAN 4.7123889
```

This would decrease both execution time (since expressions would not have to be evaluated) and memory requirements (because space would not be required to perform calculation).

A recommended addition to the COSAGE PREAMBLE is the definition of several real global variables. Variables identified to this point are:

```
TEN.16THS  
16.TENTHS
```

These variables would then need to be set to 10/16 and 16/10 respectively in the COSAGE source code.



## 6.0 SUMMARY

SAI has conducted a study of the COSAGE model. The focus of this study was to identify fruitful areas for COSAGE optimization which would reduce memory requirements and/or execution time.

In order to accomplish this, SAI applied various analysis tools and techniques to the COSAGE program. These tools and techniques included:

- Processing the COSAGE SIMSCRIPT Source Code with SAI-SDDL. The results of this effort provided a standardized format for reviewing the source code. It enhanced the source code with automated indentation and program flow of control arrows. Additionally, source code summary information (i.e., table of contents, module invocation hierarchy tree, and various cross reference listings) was generated.
- Developing Input Format Specifications for the COSAGE Program. They were developed directly from the source code and included such information as the required data item name, meaningful description, unit of measure, mode, and dimensionality.





- Utilizing the System Performance Monitoring (SPM) tool to analyze COSAGE model execution at the operation system level.
- Applying metrics to obtain quantitative assessments of the complexity of the source code.
- Using the VAX SIMSCRIPT compiler to convert the SPERRY COSAGE source code to the VAX and to identify source code anomalies which the SPERRY compiler was unable to detect. The SIMSCRIPT language was also used to instrument the COSAGE source code.

Both static and dynamic analyses were performed on the COSAGE model. Static analyses included:

- Determining all places in the source code where memory was allocated (via the CREATE and RESERVE statements) and deallocated (via the DESTROY and RELEASE keywords).
- Identifying modules of considerable size. This was done for actual source code lines as well as the size of the object code (compiled source code).
- Tallying the modules most frequently invoked statically.

Dynamic analyses were:

- Accumulating the number of times each routine was invoked dynamically (during program execution).



- Determining CPU usage per simulated hour of program execution.
- Identifying the routines which accounted for highest CPU usage.
- Locating and correcting anomalies which occurred while reading the COSAGE input data as well as those which occurred during simulated time.
- Performing control complexity, Halstead length, and level of nesting metrics on the COSAGE source code.

As a result of these analyses, a variety of changes are recommended. They include:

- Changing the method used to accomplish exponentiation in the COSAGE model. ✓
- Replacing inefficient mathematical expressions. ✓
- Streamlining unnecessary usage of the SIMSCRIPT square root function. ✓
- Changing SCHEDULE statements to RESCHEDULE statements when appropriate. ✓
- Removing/replacing routines. Some of these routines are unused and some should be replaced by a global variable. ✓
- Utilizing the SIMSCRIPT TEXT feature to save memory and enhance COSAGE transportability. ✓

AD-A148 348

COSAGE (CONCEPTS ANALYSIS AGENCY'S COMBAT SAMPLE  
GENERATOR) ANALYSIS AND. (U) SCIENCE APPLICATIONS INC  
LA JOLLA CA D A HEIMBURGER ET AL. 29 APR 84  
MDA903-83-C-0424

2/2

UNCLASSIFIED

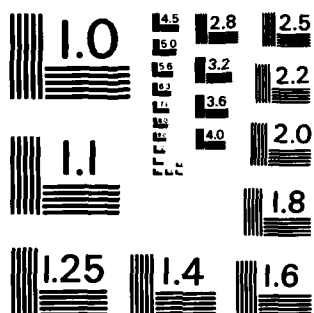
F/G 9/2

NL

END

FILED

DTIC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS - 1963 - A

- Performing a thorough analysis of the 26 most frequently invoked modules. ✓
- Modularizing identified processes to increase clarity and maintainability.
- Standardizing the COSAGE source code by developing a set of coding conventions and then applying them to the COSAGE model. ✓
- Developing graphical input/output capabilities to assist the COSAGE user. ✓
- Reorganizing the COSAGE PREAMBLE in a hierarchical fashion rather than the current semi-alphabetical manner. ✓



**END**

**FILMED**

**1-85**

**DTIC**